

Modularity for Reusable Large Language Models: External PEFT Modules and Internal Circuits

Mohammed Sabry

B.Sc.

A dissertation submitted in fulfilment of the requirements for the award of Doctor
of Philosophy (PhD)

to the



School of Computing, Dublin City University

Supervisors:

Prof. Anya Belz, Dublin City University

Prof. Suzanne Little, Dublin City University

August 2026

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy (PhD) is entirely my own work, and that I have exercised reasonable care to ensure that the work is original and have conformed to the regulations on the use and declaration of Generative AI, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: Mohammed Sabry

ID No.: A22270058

Date: 26 August 2026

Generative AI Declaration

In accordance with DCU’s guidance on declaring the use of Generative Artificial Intelligence (Gen AI) tools in postgraduate research, I report below the tools, outputs, prompts, iterations, and critical reflection associated with my use of Gen AI in preparing this thesis.

Tools (T): I used ChatGPT, including GPT-5-family models, during the drafting and revision stages of this thesis. For the final formatting and compliance review on 26 August 2026, the model used was GPT-5.6 Sol. Earlier model versions varied with availability in ChatGPT and are therefore reported at the GPT-5-family level where an exact version record was not retained. The tool was used to support editorial refinement of the written presentation of the work.

Outputs (O): The outputs generated and used were editorial in nature. These included suggestions for improving sentence flow, reducing awkward phrasing, checking grammar and clarity, strengthening transitions between paragraphs, and assessing whether the tone and narrative were consistent across chapters. I also used the tool to check whether particular passages were explained at an appropriate level for readers from different technical backgrounds. The tool was not used to generate research results, fabricate citations, produce analysis in place of my own, or supply unverified factual content for inclusion without review.

Prompts (P): Prompts were typically short, section-specific editorial questions asked on draft thesis text pasted in small portions. Representative examples include:

- “What do you think about this paragraph?”
- “Is the narrative clear here?”
- “Is the cognitive load for the reader too high?”
- “Is the thesis tone aligned across these chapters?”
- “Improve the flow of this sentence while preserving the meaning.”

In most cases, I submitted my own draft text and asked for feedback on clarity, tone, structure, or readability rather than requesting new substantive content.

Iterations (I): Use was iterative and human-directed. I typically submitted a draft passage, reviewed the suggested rewording or feedback, accepted only selected changes, and then reread the resulting text to ensure that it accurately reflected my intended meaning. In many cases, this involved one or more follow-up prompts refining wording, level of explanation, or coherence with surrounding sections.

Critical Reflection (C): Gen AI was used only as an editorial support tool and not as a substitute for scholarly judgment. All outputs were critically reviewed by me and were accepted, modified, or discarded based on whether they preserved my intended meaning, matched the technical content accurately, and remained consistent with the argument of the thesis. The conceptual framing, experimental design, interpretation of results, synthesis, and overall argumentation are my own. I take full responsibility for the final text as submitted.

Acknowledgements

This thesis marks the end of an important journey in my life, and I am deeply grateful to the people who made it possible.

First, I want to thank my family: my mother, my father, and my brother. Their support, patience, and belief in me sustained me throughout this PhD, especially during the more difficult periods. No matter how uncertain or demanding the process felt, I always knew I could rely on their encouragement. I owe them more than I can express in words.

I am also very grateful to my PhD colleagues and to the NLG group for the intellectual community, support, and shared experience they provided over the years. Research can often feel solitary, but the conversations, feedback, and day-to-day camaraderie around me made this journey much richer and more meaningful. I benefited greatly from being part of a group that cared not only about good results, but also about thoughtful discussion and curiosity.

I am especially grateful to my supervisors, and in particular to Prof. Anya Belz, for contributing so much to the way I now think about research. I began this journey thinking primarily like an engineer. Since my undergraduate years, I had been drawn to building systems, running large experiments, and making code and pipelines work at scale. Those instincts remain an important part of how I work, and they shaped much of the practical side of this thesis. But over time, this process taught me to think more fully like a scientist. I learned that research is not only about getting experiments to run, but also about asking the right questions, validating claims carefully, writing clearly, presenting ideas effectively, and communicating them at the right level for different audiences. Earlier on, I did not value these parts of the process enough. I now see them as central to doing meaningful research.

For that growth, I am grateful not only to the formal structure of the PhD, but also to the many people around me who pushed me to refine my thinking, explain my work more clearly, and hold my ideas to a higher standard. This thesis reflects not just the results I obtained, but also the researcher I became while producing it.

I also gratefully acknowledge the support of the ADAPT Centre and the funding structures that made this research possible.

Finally, I am thankful to everyone who, in one way or another, contributed to this journey. Completing a PhD is never the work of one person alone.

Contents

List of Abbreviations	10
List of Figures	13
List of Tables	16
List of Publications	20
 I Foundations	 22
1 Introduction	23
1.1 Thesis topic	23
1.2 Conceptual foundation: Modularity for reuse under constraints . .	25
1.3 Research questions	27
1.4 Thesis structure	28
1.5 Research outputs and impact	29
1.5.1 Publications and dissemination	29
1.5.2 Open-source model releases	30
 2 Background	 31
2.1 Modularity	32
2.1.1 Modularity as a system property	32
2.1.2 Why modularity matters under constraints	35
2.2 Modularity in foundation models	36
2.2.1 External PEFT modules: reusable adaptation modules . .	36
2.2.2 Internal circuits: ICL, reuse without parameter updates, and load-bearing structure	40
2.3 Summary	44
 3 Research Methodology	 46
3.1 Overview: operationalising modularity as reuse under constraints	46
3.2 Common experimental scaffolding	47
3.2.1 Budgeted comparisons	47
3.2.2 Evaluation hygiene and reporting defaults	48
3.2.3 Prompting, output normalisation, and evaluation extraction	49
3.3 Methods for external PEFT modules (Part II)	50
3.3.1 Module definition and interface	50
3.3.2 Standardised module descriptions (PEFT-Ref framework) .	51

3.3.3	Portability protocol and baselines (PEFT-Port framework)	51
3.3.4	Resource organisation under budgets	52
3.3.5	Distillation as controlled behavioural transfer	53
3.4	Methods for internal circuits (Part III: ICL, emergence, and load-bearing structure)	54
3.4.1	Matched-compute data regimes	54
3.4.2	Four-axis evaluation: behavioural capability, mechanistic signatures, causal reliance, and language-model quality . .	55
3.4.3	Mechanistic signatures: operational definitions	56
3.4.4	Operationalising early emergence	56
3.4.5	Operationalising load-bearing structure via targeted ablations	57
3.5	Summary	58
 II External PEFT Modules		60
 4 PEFT-Ref: A Reference Framework for Parameter-Efficient Fine-Tuning		61
4.1	Modular PEFT reference architecture	62
4.2	Structural properties of PEFT modules	64
4.2.1	Positioning relative to prior work	68
4.3	Characterising PEFT techniques with PEFT-Ref	70
4.3.1	Prompt-Tuning	71
4.3.2	Prefix-Tuning	71
4.3.3	LoRA	72
4.3.4	Houlsby-style Adapters	73
4.3.5	Tiny-Attention Adapters	73
4.3.6	Compacters	74
4.3.7	(IA) ³	74
4.4	Efficiency and performance comparisons with PEFT-Ref	75
4.4.1	Efficiency trade-offs under the PEFT-Ref lens	75
4.4.2	Representative task-performance evidence	78
4.5	Implications of PEFT-Ref for module design and thesis progression	80
4.5.1	Design implications for selecting PEFT modules	80
4.5.2	Design hypotheses and future module directions	82
4.6	Summary and thesis linkage	84
 5 PEFT-Port: On the Portability of Parameter-Efficient Fine-Tuning Modules		86
5.1	Why portability matters for PEFT as reusable modules	87
5.2	Preliminaries: portability protocol, controls, and operational criteria	88
5.2.1	PEFT-Port protocol and comparison conditions	88
5.2.2	Operational definitions and reported measures	89
5.2.3	Positioning relative to prior work	91
5.3	Experimental design and common scaffolding	92
5.3.1	Host models	93
5.3.2	Tasks and datasets	94

5.3.3	PEFT techniques	95
5.3.4	Budgets, steps, and seeds	97
5.4	Controlled-baseline study: testing favourable initialisation	97
5.4.1	Study-specific experimental set-up	97
5.4.2	Aggregate results	98
5.4.3	Porting-direction effects	98
5.5	Full portability study: transfer across tasks, models, and budgets	101
5.5.1	Study-specific experimental set-up	101
5.5.2	Descriptive overview of the full experimental grid	102
5.5.3	Zero-shot non-collapse rates	107
5.5.4	Matched-budget portability criteria	108
5.5.5	Pretrained task knowledge and porting direction	117
5.6	Discussion: What governs portability?	118
5.6.1	Host- and task-level factors	118
5.6.2	Module-level factors through the PEFT-Ref lens	119
5.7	Implications for budgeted adaptation in Chapter 6	120
5.8	Summary	120
6	Budgeted Adaptation: Constraining LoRA Components via Budget-Constrained Optimisation	122
6.1	From portability to budgeted adaptation	123
6.2	Positioning relative to prior work	125
6.3	Method: budgeted adaptation with LoRA	127
6.3.1	Student construction and distillation objective	127
6.3.2	Budget-aware gated LoRA modules	129
6.3.3	Budget scheduling and controller	130
6.3.4	Post-training compression for efficient inference	132
6.4	Experimental setup	135
6.4.1	Models, student regime, and training data	136
6.4.2	Evaluation measures and reported quantities	138
6.4.3	Function-style ICL probes	138
6.4.4	Compared methods	139
6.5	Results	140
6.5.1	Quality–efficiency trade-offs	141
6.5.2	Retention of in-context learning behaviour	142
6.6	Interpretation and analysis	144
6.6.1	Interpreting the observed operating points	144
6.6.2	Perplexity, teacher scale, and ICL retention	144
6.6.3	Budget-dependent reallocation of computation	145
6.7	Discussion	146
6.8	Implications for external PEFT modules and internal circuits . . .	147
6.8.1	Implications for PEFT as reusable adaptation modules . . .	147
6.8.2	Implications for internal circuits and behaviour-relevant structure	147
6.9	Summary	148

III Internal Circuits: In-Context Learning and Load-Bearing Structure	150
7 In-Context Learning Capability: Circuit Mechanisms & Load-Bearing Structure	151
7.1 Preliminaries	152
7.1.1 Operational definitions: signatures, circuits, and load-bearing tests	152
7.1.2 ICL as reuse without parameter updates: why circuits matter	157
7.1.3 Problem framing: signatures are not the same as load-bearing structure	159
7.1.4 Positioning relative to prior work	160
7.1.5 Chapter outline	162
7.2 Experiments	162
7.2.1 Bi-Induct: a minimal mechanism-targeted data intervention	163
7.2.2 Experimental setup and measurement	166
7.2.3 Models, data, and matched compute	167
7.2.4 Four-axis evaluation: behavioural capability, mechanistic signatures, causal reliance, and language-model quality . .	168
7.3 Results	170
7.3.1 Evaluation axis 1: Behavioural capability under matched compute	171
7.3.2 Evaluation axis 2: Mechanistic signatures through induction and anti-induction telemetry	173
7.3.3 Evaluation axis 3: Causal reliance through load-bearing head ablations	175
7.3.4 Evaluation axis 4: Language-model quality check	178
7.4 Discussion	180
7.4.1 Implications: reuse without parameter updates requires load-bearing structure	180
7.4.2 Limitations of the Bi-Induct design and causal test	180
7.4.3 Applying the four-axis protocol to mechanism-targeted synthetic-data interventions	181
7.4.4 Connection to modularity and the broader thesis	182
7.5 Chapter summary	182
8 Conclusion	184
8.1 Thesis summary	184
8.2 Answers to the research questions	185
8.3 Synthesis: Modularity for reuse under constraints	186
8.4 Limitations and future directions	189
8.5 Final remarks	190
A Research artefacts: Model cards	A-1
A.1 Overview and motivation	A-1
A.2 Model family summary	A-2
A.3 Intended use	A-2
A.4 Limitations and cautions	A-3

A.5	Repositories	A-3
B	Supplementary Material for Chapter 4 (PEFT-Ref)	B-1
B.1	PEFT-Ref supplementary tables	B-1
B.1.1	Task performance evidence from prior work	B-1
C	Supplementary Material for Chapter 5 (PEFT-Port)	C-1
C.1	Appendix roadmap	C-1
C.2	Experimental and evaluation details	C-2
C.2.1	Experimental grids and host-model pairs	C-2
C.2.2	Host models, datasets, and split policy	C-3
C.2.3	Distribution-matched initialisation control	C-3
C.2.4	Prompting, decoding, output normalisation, and metrics	C-4
C.2.5	PEFT technique hyperparameters	C-6
C.2.6	Training budgets and optimisation settings	C-7
C.2.7	Computational resources	C-8
D	Supplementary Material for Chapter 6 (Budgeted LoRA)	D-1
D.1	Ablation studies	D-1
D.1.1	Distillation hyperparameters and student initialisation	D-2
D.1.2	LoRA design choices	D-4
D.1.3	Budgeted LoRA: scheduling and compression	D-5
D.2	Training compute estimation	D-7
D.2.1	Training compute model	D-7
D.2.2	Average dense retention	D-8
D.2.3	Instantiated values	D-9
D.2.4	Summary	D-10
D.3	In-context learning capabilities	D-10
D.3.1	Function-style probes	D-10
D.3.2	Performance	D-13
E	Supplementary Material for Chapter 7 (In-Context Learning Capability)	E-1
E.1	Appendix roadmap	E-1
E.2	Supplementary evaluation details	E-2
E.2.1	Benchmarks and protocols	E-2
E.2.2	Supplementary copy-head telemetry details	E-3
E.2.3	Supplementary language-modelling quality check	E-4
E.3	Design ablations and robustness checks	E-4
E.3.1	Span-length sweep	E-5
E.3.2	Initial mix-ratio sweep	E-5
E.4	Additional results and analyses	E-9
E.4.1	In-context learning performance	E-9
E.4.2	In-context learning robustness	E-12
E.5	Supplementary head-ablation results	E-28

List of Abbreviations

(IA)³ Infused Adapter by Inhibiting and Amplifying Inner Activations.

Attn Attention.

BPE Byte-Pair Encoding.

CE Cross-Entropy loss.

EM Exact Match.

FFN Feed-Forward Network.

FLOPs Floating-Point Operations.

GQA Grouped-Query Attention.

ICL In-Context Learning.

KD Knowledge Distillation.

KL Kullback–Leibler divergence.

KV Key–Value heads.

LLM Large Language Model.

LoRA Low-Rank Adaptation.

MAC Multiply–Accumulate operations.

MD5 Message-Digest Algorithm 5 hash.

MLP Multi-Layer Perceptron.

PEFT Parameter-Efficient Fine-Tuning.

PF Prefix Tuning.

PPL Perplexity.

PT Prompt Tuning.

QKV Query/Key/Value Matrix in the Transformer neural network architecture.

RoPE Rotary Position Embeddings.

SVD Singular Value Decomposition.

List of Figures

4.1	PEFT-Ref reference architecture. The central grey box shows the repeated Transformer layers and residual flow of the host model. Dashed regions indicate insertion slots used by different PEFT techniques. The diagram is intended as a structural reference: it shows where PEFT modules interface with the host model, what parts of the computation they interact with, and how different techniques differ in insertion location and workspace.	63
5.1	Controlled-baseline results. Each chart shows average accuracy over three seeds and the Base/Large host pairs for one PEFT technique, direction, and pre-porting budget. Accuracy is plotted against post-porting steps; blue = ported, orange = distribution-matched sampled, and green = from scratch. The panels are enlarged and continued on the following page for readability. . . .	99
5.2	Performance of ported PEFT modules; black diamonds denote matched from-scratch modules. Results are averaged over two seeds. Within each task panel, rows are pre-porting budgets (5K, 10K, 20K, and 40K), columns are porting directions, and the x-axis gives post-porting steps (0, 0.5K, 1.5K, 3K, and 5K). The seven task panels are enlarged across four consecutive pages so that labels and values remain readable.	103
5.3	Recovery of exporting-side performance across pre- and post-porting budgets. Each cell is the importing-side score divided by the exporting-side score, averaged across models and two seeds. Values near 100% indicate near recovery and values above 100% exceed the exporting-side reference. The four panels are presented on separate pages at full width for readability. This panel shows same-dataset, instruction-tuned $\rightarrow$ raw transfer.	112

6.1	Budgeted adaptation with LoRA as resource organisation during distillation. Left: KD LoRA adds low-rank updates but leaves the dense backbone structurally unchanged at inference. Middle: the proposed method introduces a global dense-compute budget, realised through module-level dense-retention coefficients together with learned rank gates, so that computation is explicitly allocated between retained dense pathways and low-rank pathways during training. Right: after training, the student is consolidated into a deployment-efficient form by removing, approximating, or retaining dense components according to the resulting allocation profile.	128
6.2	Macro-averaged ICL composite score over the 19 function-style tasks for students distilled from teachers at three scales (0.13B, 0.5B, and 1B). Bars show the mean across tasks and three random demonstration seeds, with error bars indicating variability across demonstration seeds. The dashed horizontal line marks the ICL composite score of the full 0.13B model used as the student initialisation source.	143
7.1	Examples of copy-style snippets used in the Bi-Induct replacement-based data intervention. Each snippet is a span of L random non-special tokens, followed by a separator, then either the same span (induction) or the reversed span (anti-induction). Colours align repeated tokens across the two halves. The illustration uses $L=5$ for clarity.	166
7.2	ICL composite (macro) across two evaluation families: (a) Standard LM benchmarks; (b) Todd et al. (2024)’s function-probe suite. Each panel groups by model size (0.13B, 0.5B, 1B), with bar colours identifying the training regime (Baseline, Induction, Anti, Balanced); error bars show ± 1 standard deviation. For per-task results see Appendix E.4.1, Table E.4.	173
7.3	Layer-wise copy-head telemetry. Top row: induction scores; bottom row: anti-induction scores. For each layer I plot the best-scoring head (top 2% by score with a floor of one head per layer), averaged over three seeds, for the 0.13B, 0.5B, and 1B models. For head counts for each model see Table 6.1 (Chapter 6).	176
E.1	Layer-wise copy-head telemetry. Top row: induction scores; bottom row: anti-induction scores. For each layer I plot the best-scoring head (top 2% by score with a floor of one head per layer), averaged over six seeds, for the 0.13B model with initial mix ratios: 25%, 50%, and 100%. Head counts for each model are given in Table 6.1.	E-9
E.2	Sensitivity of ICL composite (macro) to the number of shots across two evaluation families: (a) standard LM benchmarks (3-shot vs. 1-shot); (b) Function-probe suite of Todd et al. (2024) (10-shot vs. 1-shot). Each panel groups models by size (0.13B, 0.5B, 1B), colours the bars by regime (Baseline, Induction, Anti, Balanced), and shows ± 1 standard deviation error bars.	E-21

E.3	Function-probe suite of Todd et al. (2024) - ICL Composite (macro). Three panels (0.13B, 0.5B, 1B). For each model, bars compare <i>ICL (default - no label shuffle)</i> vs <i>ICL (label shuffle)</i> across four regimes (Baseline, Induction, Anti, Balanced). Error bars show ± 1 standard deviation	E-22
E.4	Function-probe suite of Todd et al. (2024) - ICL Composite (macro). Three panels (0.13B, 0.5B, 1B). For each model, bars compare <i>ICL (HITS@1)</i> vs <i>ICL (HITS@3)</i> across four regimes (Baseline, Induction, Anti, Balanced). Error bars show ± 1 standard deviation	E-25
E.5	Function-probe ICL composite under clean evaluation, selected induction-head ablation, and matched random-head ablation. The three panels show 0.13B, 0.5B, and 1B models across Baseline, Induction, Anti-Induction, and Balanced curricula; error bars are ± 1 standard deviation. The landscape layout enlarges labels and error bars for readability.	E-30

List of Tables

4.1	PEFT-Ref structural characterisation of seven widely used PEFT techniques. The table summarises the property values defined in Section 4.2 and serves as the reference point for the technique-level discussion in Section 4.3.	69
4.2	Representative module-compute and parameter-footprint comparison for seven PEFT techniques. Here d_m is the model dimension; d_h is the module or bottleneck dimension where relevant; n is prompt or prefix length; r is LoRA rank; T is the input sequence length; and N is the reduction dimension in Compacters. Here, $\mathcal{O}(1)$ denotes modules whose output can be produced in a single step from their parameters, for example through a fixed-length embedding lookup or directly learned parameter vectors.	76
4.3	Representative comparative findings brought into the main body from the literature surveyed in Appendix B.1.1. The fuller paper-by-paper collation remains in the appendix; the present table is intended to support the chapter’s qualitative comparative claims.	79
5.1	Corresponding raw and instruction-tuned host models used in PEFT-Port. Porting is performed only between same-scale host pairs.	94
5.2	Task families and datasets used in the PEFT-Port study.	95
5.3	Structural properties of the PEFT techniques used in this chapter, expressed using the PEFT-Ref typology introduced in Chapter 4. The final column summarises the basic architectural compatibility required to import a module without changing its interface. <i>Note:</i> Compacter uses a Kronecker-product reparameterisation of the Houlsby-style Adapters MLP.	96
5.4	Mean accuracy (variance) for ported, sampled (distribution-matched), and from-scratch scenarios on sentiment analysis. Values are aggregated over exporting/importing model pairs (Base/Large; raw/Flan), pre-/post-porting step budgets, and random seeds. RT denotes Rotten Tomatoes.	101
5.5	Rates of zero-shot non-collapse across models, tasks, and budgets.	108
5.6	Percentage of cases where ported modules outperform from-scratch modules under matched post-porting budgets.	110
5.7	Percentage of cases where ported modules recover at least 99% of exporting-side performance after post-porting adaptation.	111

5.8	Percentage of cases where ported modules both outperform from-scratch modules and recover at least 99% of exporting-side performance.	117
6.1	Teacher-model presets used in this chapter. All teachers follow the same compute-optimal pretraining setup, while the deployment student is a fixed 6-layer model distilled from the selected teacher. <i>Hidden size</i> denotes the Transformer residual width, <i>MLP width</i> the feed-forward expansion dimension, <i>Attention heads / KV heads</i> the number of query heads and grouped key-value heads, and <i>Val. PPL</i> the validation perplexity on the held-out THE PILE slice.	137
6.2	Quality-cost comparisons across distillation methods. Every quality entry evaluates the same fixed 6-layer deployment-student architecture; the three columns vary <i>teacher scale</i> , not deployed model size. Training compute is an adapted-module proxy normalised to KD Full and excludes teacher cost. Inference speedup and parameter reduction cover the replaced/compressed modules only, not the full student.	141
6.3	Post-training compression statistics for Budgeted LoRA across budgets. Speedups and parameter reduction are measured over the replaced/compressed modules only.	146
7.1	Percent change in function-probe ICL composite after ablating either top-2% induction heads per layer (Δ_{induct}) or a matched number of random heads (Δ_{rand}), relative to the clean run. Negative is worse.	178
7.2	Held-out perplexity (PPL $\downarrow$) on the fixed THE PILE eval split at iso-FLOPs. Values are averaged over three seeds. For each model size, synthetic curricula use a 50% initial mix ratio linearly annealed over the full training budget.	179
A.1	Released checkpoints in the Bi-Induct model family on Hugging Face. The family spans three scales (125M, 500M, 1B) and four training regimes per scale (Baseline, Induction, Anti-Induction, Balanced), providing controlled research artefacts for reproducible experiments on in-context learning, mechanistic analysis, distillation, and modular adaptation.	A-3
B.1	Average PEFT technique accuracies on different tasks across datasets, as reported in prior comparative studies. Tiny-Attention Adapters are omitted due to the absence of comparative studies in the published literature. To preserve the required minimum font size, study context, training details, and reported accuracies are presented in separate parts.	B-2
C.1	Summary of the two PEFT-Port experimental grids. Zero denotes immediate evaluation after import, without post-porting updates.	C-2

C.2	Prompt formats, target outputs, and maximum sequence lengths used in the PEFT-Port evaluation code. Maximum lengths are measured in tokenizer tokens.	C-5
D.1	The Pile Validation perplexity (PPL) for different distillation hyperparameter settings, comparing warmstart and scratch initialisation. Lower is better.	D-3
D.2	The Pile Validation perplexity (PPL) for different student build modes under warmstart KD Full with fixed distillation hyperparameters ($T=2.0$, $\lambda_{KD}=0.5$). Lower is better.	D-3
D.3	The Pile Validation perplexity (PPL) for warmstart KD Full under two student build modes, mixed and middle, across different distillation hyperparameter settings. Lower is better.	D-4
D.4	The Pile Validation perplexity (PPL) for warmstart KD LoRA across different LoRA ranks under fixed distillation settings ($T=3.0$, $\lambda_{KD}=0.8$) and mixed student build mode. Lower is better.	D-4
D.5	Effect of budget scheduling ($F=0.0$).	D-6
D.6	Tasks, evaluation metrics, and shot counts used to compute the ICL composite in Section 6.5.2. Here HITS@1 accuracy corresponds to exact-match correctness of the first generated answer after the task-level canonicalisation used by the evaluation script.	D-12
D.7	Per-task in-context learning (ICL) performance on 19 Todd et al. (2024) function-style tasks. I report mean $\pm$ standard deviation across runs. Columns group methods (Full model, KD Full, KD LoRA, and Budgeted LoRA) across teacher scales (0.13B, 0.5B, 1B). The final row shows the macro-averaged ICL composite score used in Figure 6.2. At the larger teacher scales, the budgeted variants have higher means than KD LoRA on many, but not all, tasks; this comparison is specific to the reported setup.	D-14
E.1	Benchmarks, evaluation metrics, and shot counts used to compute the behavioural ICL composite in Section 7.3.1.	E-2
E.2	Span-length sweep at 0.13B on THE PILE. All curricula are linearly annealed over the full training budget with initial mix of 25%. Results are averaged over six seeds. Evaluation is 5-shot. I report per-task accuracies, the macro ICL composite, and held-out perplexity (PPL).	E-6
E.3	Initial mix-ratio sweep at 0.13B on THE PILE. All curricula are linearly annealed over the full training budget with span fixed at $L = 20$. Results are averaged over six seeds. Evaluation is 5-shot. I report per-task accuracies, the macro ICL composite, and held-out perplexity (PPL).	E-10
E.4	Results across model scales (0.13B, 0.5B, 1B) on THE PILE at iso-FLOPs. Copy snippets use span $L=20$. Evaluation is few-shot: 3-shot for standard LM benchmarks, and 10-shot for function-style probes. I report per-task accuracy (or EM where standard), averaged over three seeds, and the ICL composite (macro-average across tasks). Higher is better.	E-13

E.5	Results across model scales (0.13B, 0.5B, 1B) on THE PILE at iso-FLOPs. Copy snippets use span $L=20$. Evaluation is few-shot: 1-shot for both standard LM benchmarks, and function-style probes. I report per-task accuracy (or EM where standard), averaged over three seeds, and the ICL composite (macro-average across tasks). Higher is better.	E-17
E.6	Function-probe suite of Todd et al. (2024) under label-permutation stress (Todd et al. (2024) suite): HITS@1 accuracy on 10-shot prompts with demonstration labels randomly permuted; reported as mean $\pm$ std across three seeds for 0.13B, 0.5B, and 1B, comparing Baseline, Induction, Anti, and Balanced curricula.	E-23
E.7	Function-probe suite of Todd et al. (2024) under decision-rule sensitivity: HITS@3 accuracy on 10-shot prompts, reported as mean $\pm$ std across three seeds, for 0.13B, 0.5B, and 1B, comparing Baseline, Induction, Anti, and Balanced curricula.	E-26
E.8	Function-probe suite of Todd et al. (2024) (10-shot HITS@1): per-task accuracy (mean $\pm$ std over three seeds) for Baseline curriculum. I ablate the top-2% induction heads ($\downarrow$ Induct Hd 2%) or an equal number of random heads ($\downarrow$ Rand).	E-31
E.9	Function-probe suite of Todd et al. (2024) (10-shot HITS@1): per-task accuracy (mean $\pm$ std over three seeds) for Induction curriculum. I ablate the top-2% induction heads ($\downarrow$ Induct Hd 2%) or an equal number of random heads ($\downarrow$ Rand).	E-33
E.10	Function-probe suite of Todd et al. (2024) (10-shot HITS@1): per-task accuracy (mean $\pm$ std over three seeds) for Anti-Induction curriculum. I ablate the top-2% induction heads ($\downarrow$ Induct Hd 2%) or an equal number of random heads ($\downarrow$ Rand).	E-35
E.11	Function-probe suite of Todd et al. (2024) (10-shot HITS@1): per-task accuracy (mean $\pm$ std over three seeds) for Balanced curriculum. I ablate the top-2% induction heads ($\downarrow$ Induct Hd 2%) or an equal number of random heads ($\downarrow$ Rand).	E-37

List of Publications

The following publications are based on work undertaken during my period of registration and are related to the research programme developed in this thesis:

- (Sabry and Belz, 2023) *PEFT-Ref: A Modular Reference Architecture and Typology for Parameter-Efficient Finetuning Techniques*
- (Mille et al., 2024) *DCU-NLG-Small at the GEM'24 Data-to-Text Task: Rule-based generation and post-processing with T5-Base*
- (Sabry and Belz, 2024) *Assessing the Portability of Parameter Matrices Trained by Parameter-Efficient Finetuning Methods*
- (Sabry and Belz, 2026a) *Budgeted LoRA: Distillation as Structured Compute Allocation for Efficient Inference*
- (Sabry and Belz, 2026b) *Induction Signatures Are Not Enough: A Matched-Compute Study of Load-Bearing Structure in In-Context Learning*

Modularity for Reusable Large Language Models: External PEFT Modules and Internal Circuits

Mohammed Sabry

Abstract

Large language models (LLMs) are powerful, but costly to train, adapt, and deploy when treated as monolithic systems. This thesis studies reusability and efficiency under explicit constraints on parameters, floating-point operations (FLOPs), memory, data, and deployment cost. It treats modularity as an empirical perspective on reuse under constraints: preserving, transferring, or selectively invoking useful behaviour. The thesis develops this perspective through two sets of experiments. The first concerns external parameter-efficient fine-tuning (PEFT) modules. I present a reference architecture and typology for describing PEFT methods as comparable artefacts, and I evaluate portability between corresponding same-scale T5 v1.1 and Flan-T5 checkpoints that share an architectural and training lineage. In this controlled setting, transferred modules sometimes retain useful task-specific information, but the result depends on the PEFT method, porting direction, task, and post-transfer budget. These experiments do not establish portability between independently trained same-architecture hosts, whose parameter spaces need not be aligned. I then study Budgeted LoRA, which organises dense retention and low-rank capacity under a shared budget. Its cost-only controller produces sampled quality–cost operating points within the studied distillation setup; these points are not claimed to be Pareto-optimal, and the larger-teacher budgeted students do not beat the smaller-teacher fully distilled student in absolute perplexity. The second set of experiments concerns internal computation, especially in-context learning (ICL) without parameter updates. Using Bi-Induct, a synthetic-data intervention, I test whether induction-like signatures can be strengthened under matched compute. The signatures become more visible, but targeted ablations do not show a corresponding reliable improvement in few-shot ICL. Within the settings studied, the results therefore distinguish structural or correlational evidence of reuse from evidence that a component is causally important for behaviour.

Part I

Foundations

Chapter 1

Introduction

1.1 Thesis topic

Large neural network models—and in particular large language models (LLMs)—have become central to contemporary generative AI. This progress has been enabled by three reinforcing pillars: (i) large-scale datasets, (ii) increasing compute, and (iii) the Transformer architecture. As these models have become more capable, research effort has increasingly concentrated around two broad objectives. The first is to extend the frontier of capability by improving the quality and scale of data, hardware, and training/inference efficiency. The second is to align highly capable models with human goals, making their behaviour controllable and reliable through post-training and instruction-following methods. A central challenge shared by both objectives is that modern models are expensive to train and expensive to adapt if treated as monolithic systems.

This thesis therefore focuses on **efficiency through reusability**: how to reuse a pretrained model across tasks, domains, and deployment settings without repeatedly paying the full cost of retraining or full fine-tuning. The core methodological lens is **modularity**: decomposing adaptation and computation into specialised reusable units with defined interfaces, so that useful behaviour can be preserved, transferred, or selectively invoked under constraints. In this thesis, those constraints include trainable-parameter count, rank, FLOPs, memory, training data exposure, and de-

ployment cost. Practically, this means asking when useful behaviour can be carried by a detachable adaptation component, when it can be preserved under a smaller adaptation budget, and when it can be invoked from context without changing the model weights. Chapter 2 formalises this terminology and situates it in the relevant literature.

In LLMs, reuse takes two broad forms that motivate the empirical structure of the thesis. The first is reuse through *external parameter-efficient fine-tuning (PEFT) modules*: when parameter changes are required, a frozen host model can be adapted by attaching small trainable components such as Houlsby-style adapter modules (Houlsby et al., 2019) or Low-Rank Adaptation (LoRA) modules (Hu et al., 2021). These methods make adaptation efficient by turning it into the manipulation of smaller adaptation artefacts that can, in principle, be stored, transferred, compared, and reused, rather than the rewriting of the whole model. The second is reuse through *internal circuits*: many behaviours can be elicited without changing model weights, through prompting and **in-context learning (ICL)**. In that case, reuse depends on internal computational structure—specialised circuits (Olah et al., 2020a) that may support behaviour and whose importance must be tested through intervention.

The first empirical perspective in this thesis studies **external PEFT modules**: reusable PEFT components that can be described, transferred, and organised under budgets. This perspective is developed in Part II, especially Chapters 4–6. The second empirical perspective studies **internal circuits**: reusable internal computation, with particular emphasis on which internal structures are behaviourally relied upon and which reflect redundant capacity. This perspective is developed in Part III, primarily in Chapter 7.

The empirical chapters study these two perspectives separately. They do not attempt to identify a direct causal mapping from particular PEFT modules to particular internal circuits, or to show that one gives rise to the other. Establishing such a mapping would require experiments that track how a specific external adap-

tation changes specific internal computations, which is beyond the scope of this thesis. Instead, the thesis treats external PEFT modules and internal circuits as complementary cases of the same broader problem: how reusable capability can be preserved, transferred, or invoked under explicit constraints. Their implications are synthesised in Chapter 8.

1.2 Conceptual foundation: Modularity for reuse under constraints

This thesis adopts a practical notion of modularity tailored to foundation models: *modularity is valuable insofar as it enables reuse under constraints*. The aim is not to treat modularity as an abstract architectural virtue, but as a way of organising scarce resources around structures that carry useful behaviour. From this perspective, the central question is not simply how to build capable models, but how to preserve, transfer, and selectively invoke capability once it has been learned. I formalise this perspective through three tightly coupled axes of reuse. The discussion here is intentionally thesis-level and intuitive; Chapter 2 formalises the terminology and provides the literature grounding for adaptation modules, circuits, and the modularity properties used throughout the remainder of the thesis.

The three axes of reuse are:

- **Reuse through external PEFT modules.** PEFT methods insert small trainable adaptation components into a frozen model. In this thesis, I treat these components as *modules*: explicit reusable units that can be stored, transferred, compared, and budgeted. PEFT is therefore not only an efficiency technique; it also creates detachable adaptation artefacts that can support reuse without full retraining. This perspective is developed most directly in Chapters 4 and 5.
- **Reuse through internal circuits.** LLMs also support reuse without changing model weights, most clearly through prompting and in-context learning.

This form of reuse depends on specialised internal substructures that may support behaviour at inference time. I therefore study internal circuits as candidate reusable structures whose modular status is not assumed in advance, but evaluated through mechanistic signatures and intervention-based tests. The emphasis is on the distinction between *load-bearing structure*—computations that are important for a capability—and *redundant capacity*—structure that can be removed, suppressed, or rerouted with relatively small loss of performance. Training dynamics such as emergence patterns are relevant here insofar as they help explain which internal structures later become behaviourally central. This perspective is developed in Chapter 7.

- **Reuse under budgets.** Neither PEFT-module reuse nor internal-circuit reuse is interesting in the abstract if resources are unconstrained. In practice, the relevant question is always which adaptation components, computational pathways, or internal structures should be prioritised when parameters, rank, FLOPs, memory, training data exposure, or deployment cost are limited. The budgeted-adaptation contribution in Chapter 6 and the matched-compute ICL analysis in Chapter 7 provide the main empirical tests of this perspective.

These three axes are not separate topics, but different views of the same underlying challenge. In the PEFT-module reuse case, the question is how reusable modules should be defined, transferred, and budgeted. In the internal-circuit reuse case, the question is which computations are central for reuse without parameter updates, and which are only weakly or redundantly involved. Across both, the unifying challenge is how scarce resources should be organised around structures that carry useful behaviour.

This thesis is guided by the following connective narrative: *identify what is central for capability, localise what is redundant, and allocate limited adaptation or computation accordingly*. This should not be read as a claim that every chapter instantiates the same logic in the same way. Rather, it is a synthesis that emerges across the thesis. On the external PEFT-module side, Chapters 4 and 5 first estab-

lish the objects of analysis—what reusable modules are, how they can be described, and when their functionality transfers—before Chapter 6 asks how adaptation capacity can be organised under explicit constraints. On the internal-circuit side, Chapter 7 asks which visible mechanisms are behaviourally relied upon and which are not. The job of this connective narrative is therefore to link the different chapters into one account of reuse under constraints, not to imply that each chapter independently solves the full challenge of identifying centrality, redundancy, and allocation.

The conceptual framework above turns the broad argument that modularity enables reuse under constraints into a more concrete research programme. It motivates the following research questions.

1.3 Research questions

Guided by the lens above, the thesis addresses the following research questions:

1. **Can the parameters added by PEFT methods be treated as structurally and/or functionally modular?**
 - (a) How can PEFT components be described as reusable modules with explicit boundaries, interfaces, workspaces, and integration forms?
 - (b) What evaluation procedures make PEFT-module reuse measurable and comparable across settings?
2. **Can PEFT modules be reused within new host models, and what limits their portability?**
 - (a) To what extent does task-relevant functionality carried by PEFT modules survive transfer across closely related host models and task/dataset settings?
 - (b) Which factors govern portability, and what failure modes arise when modules are reused outside their original training conditions?

3. How can adaptation resources be organised under explicit budgets?

- (a) Given deployment constraints such as parameters, rank, FLOPs, or memory, how can adaptation computation be organised across reusable modules and pathways?
- (b) In a fixed student setting, what quality–cost operating points arise when a shared dense-retention budget changes the balance between dense and low-rank pathways relative to uniform low-rank adaptation?

4. How can internal structure supporting in-context learning (ICL) be identified under matched compute?

- (a) How can load-bearing internal structure for ICL be distinguished from visible mechanistic signatures?
- (b) Under matched compute, how do different training-data conditions affect behavioural capability, mechanistic signatures, causal reliance, and language-model quality?

1.4 Thesis structure

The thesis is organised so that each part develops one component of the account of reuse under constraints.

- **Part I: Foundations.** Chapter 2 defines the conceptual vocabulary of the thesis, reviews the literature relevant to external PEFT-module reuse and internal-circuit reuse, and situates the thesis contributions within the main gaps and open questions in that literature. Chapter 3 then provides the shared methodological scaffolding used throughout, including matched-budget comparisons, module-level evaluation, and intervention-based tests of behavioural reliance.
- **Part II: External PEFT modules.** Chapter 4 introduces *PEFT-Ref*, a reference framework for describing and comparing PEFT methods as reusable

artefacts with explicit interfaces and structural properties. Chapter 5 turns from structural description to functional reuse, asking when PEFT modules transfer across host models, domains, and tasks, and what limits that portability. Chapter 6 then studies how a shared dense-retention budget changes quality and cost proxies across dense and low-rank pathways in a fixed LoRA-distillation setup.

- **Part III: Internal circuits.** Chapter 7 examines in-context learning as a form of internal reuse without parameter updates. Rather than equating the presence of mechanistic signatures with capability, the chapter distinguishes between circuits that merely exhibit interpretable activity and circuits that are load-bearing for behaviour under intervention. In this way, Part III develops the argument that internal-circuit reuse should be understood in terms of causal reliance and redundancy, not only activation patterns or descriptive signatures.

Chapter 8 synthesises the findings across these parts. It treats external PEFT modules and internal circuits as complementary empirical perspectives on reuse under constraints. The conclusion summarises the conditions under which PEFT modules were described, transferred, and budgeted, and distinguishes circuit claims by intervention evidence rather than signature visibility alone.

1.5 Research outputs and impact

1.5.1 Publications and dissemination

Parts of the work developed in this thesis have been disseminated through peer-reviewed publications, workshop submissions, and preprints during the development of the broader research programme:

- (Sabry and Belz, 2023) *PEFT-Ref: A Modular Reference Architecture and Typology for Parameter-Efficient Finetuning Techniques (Chapter 4)*

-
- *(Mille et al., 2024) DCU-NLG-Small at the GEM’24 Data-to-Text Task: Rule-based generation and post-processing with T5-Base (Chapter 4)*
 - *(Sabry and Belz, 2024) Assessing the Portability of Parameter Matrices Trained by Parameter-Efficient Finetuning Methods (Chapter 5)*
 - *(Sabry and Belz, 2026a) Budgeted LoRA: Distillation as Structured Compute Allocation for Efficient Inference (Chapter 6)*
 - *(Sabry and Belz, 2026b) Induction Signatures Are Not Enough: A Matched-Compute Study of Load-Bearing Structure in In-Context Learning (Chapter 7)*

1.5.2 Open-source model releases

The thesis has also produced practical open-source research artefacts. In particular, I released a family of 12 decoder-only language models across three scales (125M, 500M, and 1B parameters), intended to support reproducible experimentation on modularity, distillation, in-context learning, and compute-aware training. Model-card-style documentation and repository links are provided in Appendix A.

Chapter 2

Background

In this chapter, I provide the broad conceptual and literature background for the thesis. More focused positioning with respect to closely related techniques is developed within the relevant empirical chapters (Chapters 4–7). The central lens is that modern large language models (LLMs) are powerful yet difficult to *reuse* efficiently under fixed budgets of parameters, floating-point operations (FLOPs), data, memory, and deployment cost (Bommasani et al., 2022; Kaplan et al., 2020; Hoffmann et al., 2022). In this setting, **modularity** matters not merely as a design preference, but as a practical mechanism for reuse under constraints. The core idea is that a model becomes easier to reuse when useful behaviour is concentrated into components that can be updated, transferred, analysed, or selectively invoked without requiring the whole system to be rewritten. As shown in Chapter 1, the two main empirical forms of such reuse are **external PEFT-module reuse**, where behaviour is adapted through reusable PEFT components, and **internal-circuit reuse**, where behaviour is reused through internal computation without parameter updates.

The chapter proceeds in three steps. Section 2.1 introduces modularity as a general principle, explains why it is especially relevant under resource constraints, and defines the structural and functional properties of modularity used throughout the thesis. Section 2.2 reviews the main research strands most relevant to modularity in the foundation-model era: parameter-efficient adaptation, transfer and portabil-

ity of adaptation, adaptation under budgets, in-context learning as reuse without parameter updates, mechanistic interpretability, and mechanism-targeted data interventions. Section 2.3 summarises the chapter and explains the transition to the shared methodological framework in Chapter 3.

2.1 Modularity

2.1.1 Modularity as a system property

Modularity is commonly treated as an important organisational property of complex systems (Tempero et al., 2023; Kadelka et al., 2023), but for the purposes of this thesis it needs a more explicit and operational formulation.

Definition. Let a system S be decomposed into components $\{m_i\}_{i=1}^k$, where each component m_i implements a function f_i that may be behavioural, computational, or parametric. Let $I(m_i, m_j)$ denote the interface between components m_i and m_j , and let G denote the system-level behaviour of S . In this thesis, *modularity* is treated as a property M of S such that:

1. each component m_i implements a sufficiently well-specified function f_i ;
2. the interfaces $I(m_i, m_j)$ are limited, describable, and low-dimensional relative to the internal state of the components they connect; and
3. the global behaviour G can be approximately predicted or reproduced from the functions $\{f_i\}$ and interfaces $\{I\}$ alone, such that

$$G \approx \text{Compose}(\{f_i\}, \{I\}).$$

Quantitatively, modularity increases when intra-component cohesion is high relative to inter-component coupling, and when replacing or adapting a component m_i with an alternative m'_i that preserves the relevant interface yields bounded change

in G (Parnas, 1972; MacCormack et al., 2007). In this sense, modularity is not merely decomposition into parts, but decomposition into parts whose internal role and external interaction structure are sufficiently stable that local change remains meaningful.

Intuition. A modular system is one that can be divided into units that do most of their own work internally and interact with the rest of the system through limited, intelligible channels. Such a system is easier to adapt, analyse, and recombine because useful behaviour is concentrated into parts rather than diffused uniformly across the whole system.

Structural and functional modularity properties. The formal definition above naturally induces two broad dimensions of modularity. **Structural properties** describe how a component is separated from the rest of the system and how it is interfaced. **Functional properties** describe what kind of reuse the component supports. These dimensions are analytically distinct but related: structural modularity determines what kind of object can be isolated or manipulated, while functional modularity determines what sort of reuse that object enables.

Structural properties. Let m_i be a component of S . In this thesis, the most relevant structural properties are:

- **Boundary:** the extent of m_i , i.e. what is inside the component and what is outside it;
- **Interface:** the signals exchanged between m_i and the rest of the system, i.e. the family of interfaces $I(m_i, \cdot)$;
- **Workspace:** the part of the host system over which m_i operates; and
- **Integration form:** the manner in which m_i contributes to global computation, for example by addition, prefixing, gating, or low-rank reparameterisation.

Definition. A component m_i is *structurally modular* to the extent that its boundary, interface, workspace, and integration form can be specified with sufficient precision that the rest of the system can interact with m_i primarily through $I(m_i, \cdot)$, without requiring direct access to its full internal state.

Intuition. Structural properties answer the question: *what kind of thing is this component, and how is it connected to the rest of the system?*

Functional properties. Functional properties describe what kinds of reuse a component supports. In this thesis, the primary functional emphasis falls on **encapsulation** and **portability**, which are defined below.

Encapsulation. Definition. A component m_i is *encapsulated* with respect to a behaviourally relevant function f_i when a substantial share of that function is concentrated within m_i , such that replacing, adapting, or removing m_i yields a bounded and intelligible change in global behaviour G without requiring coordinated redesign of the rest of S .

Intuition. Encapsulation means that the component matters *as a unit*: enough useful behaviour is concentrated inside it that local intervention has a predictable system-level effect.

Portability. Definition. A component m_i is *portable* from a source setting S to a target setting S' when it retains useful functionality after transfer, provided that the relevant interface conditions are sufficiently preserved:

$$m_i : S \rightarrow S' \quad \text{with preserved } I \Rightarrow \text{bounded useful contribution to } G'.$$

Intuition. Portability means that the component remains useful outside the exact setting in which it was trained.

2.1.2 Why modularity matters under constraints

Under fixed budgets, modularity properties turn adaptation from a global rewriting problem into a local reuse problem. If useful behaviour is sufficiently encapsulated, then only the relevant part needs to be invoked, updated, or analysed. If components are portable, then useful adaptation effort need not be repeated from scratch in every new setting. Structural modularity matters here because it determines whether such functional properties can be realised in practice: without sufficiently clear boundaries, interfaces, and integration forms, it becomes difficult to isolate, transfer, or selectively invoke the relevant part of the system. Under constraints, these properties are therefore what make modularity practically valuable rather than merely descriptively interesting.

This perspective has antecedents in several domains. In biological systems, modular organisation has been studied in neural, genetic, and protein interaction networks (Alon, 2006; Carroll, 2001; Wagner et al., 2007; Hintze and Adami, 2008). A recurrent hypothesis in these fields is that modularity tends to arise when environments vary but tasks retain reusable substructure, because such structure supports adaptation without redesign of the whole organism or network (Kashtan and Alon, 2005; Kashtan et al., 2007). In software engineering, modular decomposition supports independent development, testing, maintenance, and reuse by isolating design decisions behind interfaces (Parnas, 1972; Baldwin and Clark, 2000). Across these settings, the common idea is the same: modular structure reduces the cost of modifying, reusing, or recombining a system.

For this thesis, that general idea becomes especially important because modern LLMs are powerful but expensive to retrain, fully adapt, or redeploy. Under such conditions, the practical question is not only how to build capability, but how to preserve, transfer, and selectively invoke capability once learned. This is the sense in which constraints matter: they force attention onto the *organisation* of useful behaviour.

Position in the thesis. The structural and functional modularity properties outlined above become concrete in this thesis through two empirical forms of reuse in foundation models: **external PEFT-module reuse**, where adaptation is carried by explicit trainable modules attached to a frozen host model, and **internal-circuit reuse**, where reuse depends on internal computational structure. These two forms were introduced at the thesis level in Chapter 1; the remainder of this chapter develops the literature background for them in the specific setting of modern LLMs.

2.2 Modularity in foundation models

This section reviews the main literature strands that motivate the thesis and shows how the general modularity framework above becomes concrete in modern LLMs. The discussion is broad rather than exhaustive: the aim is to situate the thesis in the major research directions relevant to external PEFT modules and internal circuits, while leaving detailed chapter-local positioning to the empirical chapters.

2.2.1 External PEFT modules: reusable adaptation modules

As introduced in Chapter 1, modern LLMs are typically pretrained once and then adapted repeatedly to new tasks, domains, and deployment settings (Bommasani et al., 2022; Kaplan et al., 2020; Hoffmann et al., 2022). Full fine-tuning often undermines reuse because it produces a new dense parameter set for each task. Parameter-efficient fine-tuning (PEFT) methods address this by attaching or learning relatively small task-specific parameter components on top of a frozen model, including Houlby-style adapter modules (Houlby et al., 2019), prompt- and prefix-based methods (Lester et al., 2021; Li and Liang, 2021), and low-rank methods such as LoRA (Hu et al., 2021). Recent surveys make clear that PEFT is not a single technique family but a structured design space (Ding et al., 2023; Han et al., 2024; Zhang et al., 2025).

Structural properties of PEFT modules. In this thesis, a *PEFT module* is an explicit trainable adaptation component attached to an otherwise frozen host model, such that adaptation occurs primarily through the module’s own trainable parameters rather than through full modification of the host.

A first literature strand relevant here concerns frameworks that make machine-learning artefacts and evaluation claims more explicit, systematic, and comparable. Model Cards, for example, were proposed as a standardised reporting framework for trained models, with the specific aim of documenting intended use, evaluation conditions, and other properties of released model artefacts (Mitchell et al., 2018). In NLP/ML reproducibility research, Belz has similarly argued for explicit, quantitative frameworks for assessing reproducibility, so that results can be compared and interpreted under a common methodology rather than through ad hoc reporting (Belz, 2021; Belz et al., 2022). These works are not about PEFT modules specifically, but they establish an important precedent for the present thesis: if models and evaluation results benefit from standardised descriptive and assessment frameworks, then PEFT modules may also need a more explicit framework if they are to be treated as reusable artefacts rather than merely as implementation tricks.

A second strand is the PEFT survey literature itself. Existing surveys are valuable because they organise PEFT methods into families and compare their trade-offs. For example, recent surveys distinguish additive methods such as adapters, prompt- and prefix-based methods that modify conditioning, reparameterisation methods such as LoRA, and hybrid or selective approaches, while also reviewing their parameter efficiency, memory footprint, and downstream behaviour (Ding et al., 2023; Han et al., 2024; Zhang et al., 2025). However, these surveys are primarily taxonomic and algorithmic: they classify methods by mechanism and performance trade-off, but they do not usually formalise PEFT methods as reusable modular artefacts with explicit structural properties such as insertion location, interface, workspace, and integration form.

From the thesis perspective, this is the key gap. What is needed is not only a

survey or module-card-style summary, but a genuinely *modular* treatment of PEFT: one that makes clear what kind of reusable object a PEFT module is, how one PEFT method differs from another structurally, and what kinds of reuse those structural differences enable or constrain. That is the role of Chapter 4, which develops a reference framework for characterising PEFT methods as reusable modules rather than only as parameter-saving adaptation tricks.

Portability as module-level reuse. Establishing such a modular reference framework makes it possible to ask the next question relevant to the thesis: whether these modules retain useful functionality when transferred across settings. In the broader deep learning literature, *transfer* refers to reusing knowledge learned in one setting—often encoded in learned representations, features, or parameters—to improve performance in another related setting (Pan and Yang, 2010; Yosinski et al., 2014; Jiang et al., 2022). This broad transfer perspective includes feature transfer, parameter reuse, and other forms of inductive reuse across tasks or domains.

In this thesis, I use **portability** for the narrower modularity question of whether a reusable PEFT module retains useful functionality when the whole module is moved across host models, domains, or tasks. This distinction matters because portability is not simply a generic transfer-learning result: it concerns whether a module can function as a reusable artefact under controlled movement between settings. Existing work suggests that useful behaviour can sometimes survive such movement, but the evidence is often narrow and method-specific. For example, prior work studies adapter transfer across tasks or domains and transfer of LoRA modules across different base models, but much less often asks whether the transferred module itself carries reusable task-specific functionality beyond target-side optimisation advantages (Pfeiffer et al., 2020; Wang et al., 2024). Chapter 5 addresses this gap by evaluating portability directly under controlled baselines.

A related but distinct transfer mechanism is *knowledge distillation*, introduced by Hinton et al. (2015) as teacher–student transfer. Recent surveys show that distillation has expanded substantially in the era of large foundation models, including

LLM-oriented settings (Mansourian et al., 2025), and recent work has begun to study its scaling behaviour explicitly (Busbridge et al., 2025). In the thesis, however, distillation is not treated as the same object as PEFT portability: portability concerns direct reuse of an explicit PEFT module, whereas distillation concerns behavioural transfer into a new student parameterisation. For this reason, distillation becomes conceptually central in Chapter 6, where it serves as the controlled transfer setting for studying budgeted adaptation.

Resource organisation under adaptation budgets. Once PEFT modules are treated as explicit structural objects, and once portability makes clear that useful behaviour is not uniformly encapsulated across modules, workspaces, or host settings, the next question is how limited adaptation capacity should be organised under explicit deployment constraints. The issue is therefore not only whether adaptation is parameter-efficient, but how scarce capacity can be allocated so that redundant functionality is reduced while structures that carry useful behaviour are preserved.

A natural entry point for this question is the broader literature on low-dimensional and low-rank structure in deep learning. In the context of language-model fine-tuning, Aghajanyan et al. (2020) argue that successful adaptation often lives in a much lower-dimensional subspace than the full parameter space, providing empirical and theoretical support for the view that fine-tuning need not exploit all available degrees of freedom. LoRA adopts this perspective directly, hypothesising that the task-specific weight update itself has low intrinsic rank and can therefore be parameterised efficiently through a low-rank decomposition (Hu et al., 2021). More broadly, simplified theory for deep matrix factorisation shows that gradient descent can exhibit an implicit bias toward low-rank solutions, suggesting that low-rank structure can act as a useful complexity constraint, even if this does not yet amount to a complete theory of generalisation for modern LLM adaptation (Chou et al., 2023; Arora et al., 2019). Taken together, this line of work suggests that adaptation capacity need not be distributed uniformly across all available directions or modules.

A partially overlapping literature addresses the complementary question of *ca-*

capacity reduction: how model size, dense computation, or deployment cost can be reduced through *pruning* (removing unnecessary parameters or structures), *sparsification* (driving computation toward sparse use of weights or activations), *compression* (reducing representational or deployment cost), or more direct suppression of dense computation (Sanh et al., 2020b; Wang et al., 2020b; Cheng et al., 2024). This literature matters here because it shifts attention from parameter-efficient adaptation alone to the further question of which computations remain necessary once a model has been adapted. In other words, low-rank work motivates selective allocation of adaptation capacity, while pruning and compression work motivate selective removal of redundant dense capacity.

From the thesis perspective, these strands jointly motivate a broader resource-organisation question: how can the contributions of different modules and computational pathways be coordinated under explicit constraints? Chapter 6 addresses this question by studying adaptation as the organisation of dense and low-rank contributions under a shared budget, rather than treating low-rank adaptation and dense reduction as entirely separate decisions.

2.2.2 Internal circuits: ICL, reuse without parameter updates, and load-bearing structure

The second major form of modularity in modern LLMs is internal rather than external. Although Transformers are trained as dense monolithic systems (Vaswani et al., 2017), interpretability work increasingly suggests that specialised internal substructures may support particular behaviours. In the Transformer architecture, these candidate substructures are typically sought among its main computational components—for example, attention heads within multi-head self-attention, multi-layer perceptron (MLP) layers and their neuron subspaces, and the interactions among them through the residual stream. Early work on circuits and feature-level mechanisms argued that model behaviour can often be explained through structured interactions among such components rather than through undifferentiated

distributed computation alone (Olah et al., 2020b; Vig, 2019; Michel et al., 2019; Geva et al., 2021).

More recent work has extended this line through intervention-based and increasingly automated methods for identifying behaviourally relevant substructures. *Activation patching* tests whether a component is causally involved in a behaviour by replacing its activation in one run with the corresponding activation from another run and measuring the behavioural change (Wang et al., 2023; Zhang and Nanda, 2024). *Path patching* refines this idea by localising which specific computational paths between components are responsible for the behaviour, rather than only whether an individual component matters (Goldowsky-Dill et al., 2023). *Causal tracing* similarly asks where behaviourally relevant information is computed or routed by systematically intervening on internal states and tracing the effect on downstream predictions (Meng et al., 2023). Building on these tools, *automated circuit discovery* methods search for a small set of components and interactions sufficient to recover the behaviour under study, reducing the amount of manual hypothesis construction required (Conmy et al., 2023; Rai et al., 2025). Together, this literature motivates the internal side of the thesis: if internal computation is structured enough to be decomposed into behaviourally relevant subcomponents, then internal circuits become a meaningful object of study rather than a purely metaphorical one.

In this thesis, the internal-side object of study is the *circuit*: an internal specialised substructure inside an LLM that may implement an identifiable function or support behaviour. A circuit may be realised through attention heads, MLP subspaces, neuron groups, or distributed interaction patterns, but unlike PEFT modules, its boundary is usually inferred from analysis rather than specified by design. This difference is important: PEFT modules are architected as modules, whereas circuits must earn modular status through evidence. Chapter 7 therefore uses the broader circuit literature as background, but operationalises the internal object more narrowly through attention heads, which are architecturally defined and can be scored and ablated directly.

A major reason circuits matter here is that LLMs can often perform new tasks from context alone. This is the core phenomenon of **in-context learning (ICL)** (Brown et al., 2020). In this thesis, ICL is treated as a form of **reuse without parameter updates**: the model is reused for a new task through context rather than through additional optimisation. From this perspective, circuits are important because they provide candidate internal computational structures through which such reuse may be implemented.

A particularly influential line of work links ICL to induction-style mechanisms, especially *induction heads*, a specific type of attention head whose attention pattern supports copy-and-match behaviour over repeated sequences in context (Olsson et al., 2022). The technical importance of this literature is that it offers a concrete candidate mechanism for few-shot generalisation: not just the ability to copy tokens, but a structured attention motif that can exploit repeated patterns in context. For the purposes of this thesis, this matters because it gives a plausible internal target whose development and causal contribution can be studied directly.

Conceptually, however, the next question is not only whether such signatures are visible, but how far they explain ICL more generally. Subsequent work has debated whether induction heads are sufficient, whether they are developmental precursors to broader in-context learning, and how they relate to more abstract task-level generalisation (Şahin et al., 2026; Yang et al., 2026; Kahardipraja et al., 2026). From the thesis perspective, a visible mechanistic signature is therefore not enough. Once a candidate mechanism has been identified, the next question is whether the model is actually dependent on that structure for behaviour.

Encapsulation in internal structure: load-bearingness and redundancy.

On the internal side of the thesis, this dependence question is treated as a question of *encapsulation*: whether useful behaviour is sufficiently concentrated in a structure that the model relies on it under intervention. This motivates the distinction between *load-bearing structure* and *redundant capacity*. A structure is *load-bearing* when ablating or perturbing it induces a substantially larger degradation in capabil-

ity than would be expected from typical or randomly selected components. A structure exhibits *redundant capacity* when it can be perturbed, removed, or rerouted with relatively small behavioural loss, either because it is weakly used or because other components compensate.

In other words, visible internal organisation is not by itself evidence that a structure matters for reuse or capability. The key question is whether the model is behaviourally dependent on that particular internal structure. This is why causal validation becomes central. A recurring activation pattern, an interpretable motif, or an induction-like head signature is not sufficient to show that a structure is functionally important unless intervening on it produces a corresponding behavioural effect. Chapter 7 therefore places capability, mechanistic telemetry, and causal ablation in direct comparison.

A related training-dynamics question concerns *early emergence*: the possibility that some internal structures become functionally identifiable earlier in training than others. Recent work suggests that training dynamics and controlled data conditions can affect when such mechanisms appear and what subcircuits support them (Olsson et al., 2022; Singh et al., 2024). In the thesis, early emergence is not treated as an endpoint in itself, but as a possible explanatory signal. Earlier appearance of a mechanism does not by itself show that it will later be load-bearing.

This leads to a final literature strand relevant to the internal side of the thesis: synthetic data, curriculum, and mechanism-targeted interventions. Once candidate mechanisms can be identified and their behavioural importance tested, a natural next question is whether training data can be used to preferentially strengthen those mechanisms under matched compute, making them not only more visible but more central for behaviour. Scaling-law work formalises trade-offs among model size, data, and compute (Kaplan et al., 2020; Hoffmann et al., 2022), while more recent work studies how data quality, filtering, and curriculum alter learning efficiency or shift effective scaling curves (Bansal et al., 2022; Subramanyam et al., 2026). In parallel, work on algorithmic behaviour in Transformers suggests that relatively simple

primitives such as copying, search, and counting can be preferentially learned under appropriate conditions (Davidovich and Ringel, 2026). More generally, synthetic or semi-synthetic interventions have been used for at least three related purposes: to alter the order or difficulty of training examples, as in curriculum learning (Soviany et al., 2022; Bengio et al., 2009); to bias models toward particular algorithmic solution classes through procedural or synthetic data (Davidovich and Ringel, 2026; Shinnick et al., 2025); and to create controlled settings in which specific capabilities or circuits can be isolated and evaluated more directly (Gupta et al., 2024). From the thesis perspective, this makes mechanism-targeted data a natural way to test whether preferentially exciting a candidate computation under matched compute makes it not only more visible, but more behaviourally relied upon. Chapter 7 studies this question using a replacement-based synthetic-sequence intervention under matched compute (iso-FLOPs).

2.3 Summary

This chapter established the conceptual and literature background for the thesis. It first defined modularity as a system property in which useful behaviour is concentrated into components with sufficiently well-specified functions and limited interfaces, so that local change remains meaningful under fixed resource constraints. It then argued that, for the purposes of this thesis, the most important modularity properties are structural modularity on the one hand, and the functional properties of encapsulation and portability on the other, with load-bearingness later providing the thesis-specific test of whether useful internal structure is genuinely relied upon for behaviour.

The chapter next reviewed how these ideas become concrete in modern LLMs through two empirical forms of reuse. On the external PEFT-module side, it showed how the literature moves from PEFT methods as adaptation techniques, to PEFT modules as reusable structural objects, to questions of portability and, ultimately, to adaptation under budgets as a resource-organisation problem. On the internal-

circuit side, it reviewed the literature on mechanistic interpretability, ICL as reuse without parameter updates, induction-style mechanisms, and the distinction between visible internal structure and structure that is genuinely load-bearing, together with work on early emergence and mechanism-targeted data interventions under matched compute.

Taken together, this review shows that existing work provides many of the relevant ingredients for the thesis, but still leaves an important connected set of open questions about how reusable PEFT modules and reusable internal computation should be understood within a unified account of reuse under constraints. Chapter 3 therefore turns from literature background to the shared methodological framework used to study these questions across the empirical chapters.

Chapter 3

Research Methodology

3.1 Overview: operationalising modularity as reuse under constraints

This chapter provides the shared methodological framework used throughout the thesis. Its purpose is not to restate the full technical details of each later empirical chapter, but to explain how the thesis operationalises its central argument that *modularity enables reuse under constraints*. The conceptual distinctions used here—including *external PEFT-module reuse*, *internal-circuit reuse*, *encapsulation*, and *portability*—are introduced in Chapter 2; the present chapter focuses on how those concepts are translated into empirical designs, evaluation protocols, and intervention strategies across the remainder of the thesis.

The research questions in Chapter 1 are addressed through two complementary methodological programmes. The first studies **external PEFT modules**: reusable parameter-efficient fine-tuning (PEFT) components treated as explicit adaptation artefacts. This programme underlies Part II and supports the study of module definition, portability, and resource organisation under budgets. The second studies **internal circuits**: reusable computation inside the model, especially the internal structures that support in-context learning (ICL) as a form of *reuse without parameter updates*. This programme underlies Part III and supports the study of

mechanistic signatures, load-bearing structure, and the relation between behaviour and causal reliance.

Across both programmes, the methodological emphasis is the same. Comparisons should be made under controlled budgets; claims about reuse should be tied to explicit operational criteria; and descriptive or mechanistic observations should be distinguished from evidence that a system behaviourally relies on the implicated structure. In this way, the chapter turns the thesis narrative into a concrete research design.

This chapter proceeds in four stages. Section 3.2 introduces the experimental conventions shared across the thesis. Section 3.3 then describes the methodological framework used in Part II to study external PEFT modules, portability, and adaptation under budgets. Section 3.4 then describes the methodological framework used in Part III to study internal circuits, matched-compute data regimes, mechanistic signatures, early emergence, and load-bearing structure. Section 3.5 closes the chapter by summarising the shared methodological principles and linking them forward to the empirical chapters.

3.2 Common experimental scaffolding

3.2.1 Budgeted comparisons

A central thesis claim is that modular reuse should be evaluated *under constraints*. Methodologically, this means that comparisons should not confound “better reuse” with “more resources”. I therefore prioritise matched-budget designs throughout the thesis.

- **Pretraining: iso-FLOPs.** When comparing different data regimes, I hold total compute fixed in terms of floating-point operations (FLOPs) under the Chinchilla-style compute-optimal regime used in this thesis—approximately 20 training tokens per parameter (Hoffmann et al., 2022)—while varying the structure of the training data (e.g., natural-only vs. replacement-based synthetic-

sequence conditions) and keeping architecture and optimisation schedule comparable across conditions. Because the compared conditions use the same overall compute budget, differences in capability or mechanism cannot be attributed simply to spending more compute overall. Instead, they can be interpreted as evidence that the fixed compute budget has been used differently across training conditions.

- **Adaptation: explicit module budgets.** When comparing PEFT methods or constrained variants of PEFT, I report trainable-parameter budgets (e.g., LoRA rank or adapter bottleneck size), adaptation compute (steps/tokens), and, where relevant, realised inference efficiency. Comparisons are made against uniform or unconstrained baselines at the same or explicitly stated adaptation budgets, so that improved results cannot be explained simply by giving one method more trainable capacity. Instead, any gains are interpreted as evidence that the available adaptation budget has been organised more effectively across modules and pathways.

The underlying principle is that claims about modular reuse are only meaningful if they survive budget control. Otherwise, apparent gains may simply reflect larger adaptation capacity or greater training expenditure.

3.2.2 Evaluation hygiene and reporting defaults

Unless otherwise stated, I follow shared reporting conventions across empirical chapters:

- **Seeds.** Report mean and variability over multiple random seeds where feasible. Where seed counts differ across empirical chapters because of computational cost or evaluation complexity, the exact number of seeds and their role in the experiment are stated locally in the relevant chapter.
- **Prompt control.** Where evaluation depends on prompted inference, use fixed prompt templates within each experiment and average across demonstration

seeds and/or example order where relevant.

- **Checkpoint policy.** Use an explicit checkpoint-selection rule, such as the final checkpoint under matched budgets or the best held-out checkpoint where this is part of the stated design. Chapter-local deviations from this default are made explicit.
- **Language-model quality check.** Report held-out perplexity or broader language-model quality when an intervention could improve the target behaviour while degrading general modelling quality, calibration, or fluency. This acts as a safeguard against attributing gains to modular reuse when they are accompanied by hidden regressions elsewhere.

These conventions serve a methodological purpose beyond reporting consistency. They are intended to ensure that claims about modularity, reuse, or causal contribution are not driven by prompt variance, cherry-picked checkpoints, or hidden quality regressions.

3.2.3 Prompting, output normalisation, and evaluation extraction

Where later chapters involve prompted evaluation, I treat prompt construction and output extraction as part of the methodological design rather than as incidental implementation detail. Unless otherwise stated, the prompted model family, template structure, variable-slot insertion procedure, decoding settings, and answer-extraction rule are held fixed within each experiment.

Variable content is inserted into templates through an explicit slot-filling procedure appropriate to the task under study. Where multiple demonstrations are used, their content, order, and formatting are controlled so that downstream comparisons are not driven by arbitrary prompt variation. When free-form generations must be evaluated against task targets, outputs are normalised through an explicit extraction procedure appropriate to the task (for example, taking the first valid label,

canonicalised string, or structured field), so that evaluation reflects the intended target behaviour rather than formatting variance alone.

The exact prompted model families, templates, extraction rules, and chapter-specific deviations from these shared defaults are described locally where they matter most. In particular, Chapter 5 details the prompted evaluation used for module portability experiments, Chapter 6 details the prompted evaluation used to assess preservation of in-context learning capabilities under constrained adaptation, and Chapter 7 details the prompt and endpoint construction used for the internal-circuit analysis of in-context learning.

These controls ensure that later comparisons of portability, constrained adaptation, or in-context learning are not confounded by formatting artefacts or answer-extraction variance.

3.3 Methods for external PEFT modules (Part II)

3.3.1 Module definition and interface

As defined conceptually in Chapter 2, the basic unit of analysis in Part II is the *PEFT module*: the trainable adaptation component introduced on top of a frozen host model. Concretely, this may take the form of LoRA matrices, adapter multilayer-perceptron weights, or prefix/prompt parameters, depending on the PEFT method used. The host model remains fixed, while the module interacts with it through a defined insertion interface—that is, a specified location and workspace within the Transformer computation.

This definition matters methodologically because the thesis treats PEFT components not merely as optimisation tricks, but as reusable artefacts. If later claims about portability, constraint, or budgeted organisation are to be meaningful, the module must first be identifiable as a stable object of comparison: something that can be described, transferred, constrained, and evaluated as a unit.

3.3.2 Standardised module descriptions (PEFT-Ref framework)

To make module-level comparisons meaningful, I adopt the reference-framework perspective developed in Chapter 4. At minimum, each module description records:

1. insertion locations;
2. the computation workspace (for example, attention versus multilayer-perceptron subcomponents);
3. whether parameters are added directly or introduced through a low-rank or bottleneck parameterisation;
4. and any parameter sharing or tying.

This yields a standardised description of each module in terms of what trainable component is introduced, where it is inserted, what workspace it acts over, and how it contributes to the host computation. Methodologically, this matters because later comparisons then operate on explicitly specified reusable objects rather than on vague claims about “using PEFT”.

3.3.3 Portability protocol and baselines (PEFT-Port framework)

As discussed conceptually in Chapter 2, the thesis treats *portability* as a module-level property: whether a reusable PEFT module retains useful functionality when moved across host models, domains, or tasks. Methodologically, the question is operationalised through the portability framework developed in Chapter 5.

To study portability, I train a module on an *exporting* host model, import it into an *importing* host model, and measure performance under controlled post-import adaptation. To isolate transfer of learned structure from favourable initialisation, imported modules are compared against two baselines:

-
1. **From-scratch importing-side training:** the same PEFT method trained on the importing model from random initialisation, with no transferred information from the exporting-side module.
 2. **Distribution-matched initialisation:** an importing-side module initialised so that its parameters match the imported module’s first and second moments (mean and variance), without using the imported module’s actual learned parameter structure.

This protocol reflects a broader methodological principle in the thesis: claims about reuse should be tested against baselines that rule out simpler alternative explanations. Here, those alternatives include the possibility that post-import performance comes only from a favourable importing-side initialisation, or from matching coarse parameter statistics, rather than from task-relevant structure carried by the imported module itself.

3.3.4 Resource organisation under budgets

As framed in the third research question in Chapter 1, the methodological shift here is from treating low-rank adaptation primarily as a fixed parameter-efficiency technique to treating constrained adaptation as a **resource-organisation problem**. Under explicit deployment constraints, the goal is not merely to minimise trainable-parameter count, but to determine how limited adaptation capacity should be distributed across reusable modules and across the dense and low-rank pathways available within them.

In Chapter 6, this is operationalised by exposing two controllable routes inside each adapted module:

- a **dense contribution**, which determines how much of the original dense computation remains active;
- and a **low-rank contribution**, whose effective capacity is controlled through learned **rank gates**.

A global budget controller constrains total retained dense computation, while module-level retention coefficients and rank gates determine the dense and low-rank contributions allowed by the parameterisation. The methodological point is that adaptation is treated as *organising compute across reusable modules under a budget*, rather than uniformly attaching the same low-rank parameterisation everywhere.

Accordingly, Chapter 6 reports:

- achieved budget utilisation;
- allocation profiles across modules;
- realised profiles of retained dense computation and active low-rank capacity across modules;
- and the resulting quality–cost comparisons with uniform LoRA and KD Full baselines.

Together, these measurements describe how the imposed budget changes the organisation of computation within the tested setup.

3.3.5 Distillation as controlled behavioural transfer

As discussed in Chapter 2, **distillation** is treated in this thesis as a controlled form of behavioural transfer rather than as direct portability of an explicit PEFT module. In the budgeted-adaptation setting, I use distillation to study how reusable capability can be transferred from a stronger teacher into a smaller, constrained student under explicit adaptation budgets.

Methodologically, distillation is useful because it isolates the resource-organisation problem from the problem of learning the capability from scratch. Rather than asking whether a small student can independently discover the relevant behaviour, the question becomes: given that useful behaviour already exists in the teacher, how should the student’s limited adaptation capacity be organised so as to preserve it most effectively under budget?

Unless otherwise stated, the distillation objective uses a standard mixture of next-token cross-entropy and teacher–student distribution matching (Hinton et al., 2015). The full objective, its equation, and the relevant hyperparameters (including temperature, objective weights, Kullback–Leibler (KL) direction, and masking conventions) are specified in Chapter 6 where they are introduced in detail.

3.4 Methods for internal circuits (Part III: ICL, emergence, and load-bearing structure)

3.4.1 Matched-compute data regimes

As discussed conceptually in Chapter 2, the internal-side object of study in this thesis is the *circuit*: an internal computational structure that may support reuse without parameter updates. To study such structures empirically, I use a targeted replacement-based synthetic-sequence data intervention (Bi-Induct; detailed in Chapter 7) that compares natural-only pretraining with replacement-based synthetic-sequence conditions designed to exercise hypothesised in-context-learning mechanisms, especially induction-style computations that support copying over repeated sequences in context (Olsson et al., 2022). All such comparisons are conducted under matched total floating-point operations (iso-FLOPs).

This design reflects the same methodological principle used in Part II: claims about mechanism should not be confounded with differences in total resources. Because the compared conditions are matched in total compute, differences in capability, telemetry, or causal reliance can be interpreted as differences in how that fixed compute budget is used across data regimes, rather than as a consequence of simply spending more compute overall. Accordingly, the methodology treats measured circuit signatures as candidate evidence of reusable internal structure, and uses causal-reliance tests to ask whether behaviour actually depends on the implicated components.

3.4.2 Four-axis evaluation: behavioural capability, mechanistic signatures, causal reliance, and language-model quality

The central methodological logic of Chapter 7 is that internal-circuit reuse cannot be established from one kind of evidence alone. I therefore evaluate each experimental condition in the matched-compute comparison—for example, natural-only pretraining and the different replacement-based synthetic-sequence conditions—using three evidential axes plus one language-model quality check:

1. **Behavioural capability.** Does the model exhibit stronger in-context learning behaviour on the intended behavioural endpoints?
2. **Mechanistic signatures.** Do induction-like or control-aligned signatures appear in the internal computation, and where are they concentrated?
3. **Causal reliance.** Are the implicated structures load-bearing for the behaviour, or are they only correlated with it?
4. **Language-model quality check.** Does the intervention preserve general language-model quality, as measured by held-out cross-entropy or perplexity, rather than improving a target diagnostic while degrading the underlying model?

The first three axes provide evidence about internal-circuit reuse itself. **Behavioural capability** establishes whether the behaviour of interest is present at all, but behaviour alone does not reveal the mechanism producing it. **Mechanistic signatures** show whether induction-like or control-aligned patterns are present and where they are concentrated, but signatures alone do not establish that those structures are load-bearing. **Causal reliance** tests whether behaviour depends on the implicated structures, but ablation results are difficult to interpret unless there is a clear behavioural endpoint whose degradation can be measured. The fourth

axis, **language-model quality**, is a usability check: it ensures that mechanism-targeted interventions are not treated as successful if they merely amplify signatures or improve a narrow endpoint while degrading general modelling quality.

For this reason, Chapter 7 reports four corresponding result axes: behavioural capability, mechanistic signatures, causal reliance, and language-model quality. This four-axis reporting structure is consistent with the thesis methodology because quality is not an additional claim about internal-circuit modularity; it is the check that makes the matched-compute comparison interpretable.

3.4.3 Mechanistic signatures: operational definitions

Following the replacement-based Bi-Induct data regime introduced in Chapter 7, I compare matched-compute models trained under natural-only and targeted synthetic-sequence conditions designed to exercise induction-like copying behaviour. Within this framework, I use attention-head-level copy metrics to score how strongly individual heads exhibit induction- and control-aligned attention patterns; the exact metric definitions and task constructions are provided in Chapter 7. To summarise specialisation, I rank heads within each layer by these scores and report both the top 2% of heads per layer and their *mass share*, that is, the proportion of the total layer-wise score contributed by that top-scoring subset. This allows telemetry to capture not only whether induction-like signals are present, but also how concentrated they are within a small number of heads.

Methodologically, telemetry is treated as *evidence of presence and localisation*, not as proof of functional centrality. This distinction is crucial for the later argument that induction-like signatures are not sufficient evidence of load-bearing structure.

3.4.4 Operationalising early emergence

As introduced conceptually in Chapter 2, this thesis uses *early emergence* to refer to the possibility that mechanism-aligned internal signatures become functionally identifiable earlier in training than others. This usage is motivated by prior work

suggesting that the timing of induction-like or other mechanism-relevant signatures can vary across training and across data conditions (Olsson et al., 2022; Singh et al., 2024). Within the replacement-based matched-compute framework used in Chapter 7, I operationalise early emergence as the tendency for mechanism-aligned signatures to appear earlier and peak earlier across layers under matched-compute training, rather than as a claim about reaching capability in fewer optimisation steps. The exact measurements, layerwise comparisons, and chapter-specific analysis choices are detailed in Chapter 7.

In this thesis, emergence timing is therefore treated as a potentially informative signal about developmental order, but not as a substitute for direct tests of capability or causal reliance.

3.4.5 Operationalising load-bearing structure via targeted ablations

As discussed conceptually in Chapter 2, this methodology operationalises the thesis distinction between visible *mechanistic signatures* and genuinely *load-bearing*, functionally central structure. To test whether a circuit is load-bearing rather than merely active, I use targeted ablations aligned with the telemetry:

- select a per-layer quota of candidate heads (for example, the top 2% of attention heads by an induction-aligned score, with at least one head selected per layer);
- ablate those heads at evaluation and measure degradation on the relevant ICL endpoints;
- compare against ablating an equal number of randomly chosen heads per layer.

A substantially larger degradation under top-versus-random ablation indicates a more centralised, load-bearing implementation; smaller gaps are more consistent with redundancy or with more distributed implementations.

This methodology operationalises the thesis distinction between *mechanistic signatures* and *functional centrality*. A structure is not treated as important merely because it is visible in telemetry, but because targeted intervention shows that behaviour is more sensitive to removing it than to removing matched random components. Where relevant, Chapter 7 also relates these intervention results back to the matched-compute data regimes discussed above, asking whether making a candidate mechanism more visible through training data also makes behaviour rely on it more strongly.

3.5 Summary

This chapter established the shared methodological framework used throughout the thesis. Building on the conceptual vocabulary introduced in Chapter 2, it showed how the thesis operationalises modularity as reuse under constraints through common budget controls, reporting defaults, prompting and output-normalisation procedures, and explicit distinctions between descriptive evidence and causal evidence.

On the external PEFT-module side, the chapter defined the PEFT module as the basic unit of analysis, described the use of standardised module descriptions, introduced the portability protocol and its baselines, and framed constrained adaptation as a resource-organisation problem under explicit budgets. It also positioned distillation as a controlled form of behavioural transfer for studying how reusable capability can be preserved in constrained students.

On the internal-circuit side, the chapter described the matched-compute data-regime framework used to study reuse without parameter updates, the three evidential axes—behavioural capability, mechanistic signatures, and causal reliance—together with the language-model quality check used to interpret mechanism-targeted interventions, and the operational procedures used to study early emergence and load-bearing structure through targeted ablations.

Taken together, these methodological choices provide the shared experimental scaffolding for the empirical chapters that follow. Chapter 4 next begins Part II by

applying this framework to the structural description of PEFT methods as reusable modules.

Part II

External PEFT Modules

Chapter 4

PEFT-Ref: A Reference Framework for Parameter-Efficient Fine-Tuning

Parameter-efficient fine-tuning (PEFT) methods reduce the cost of full fine-tuning by attaching comparatively small trainable adaptation components to a frozen host model. In this thesis, these components are treated as *PEFT modules*: reusable artefacts that can be described, compared, transferred, and later organised under explicit adaptation budgets. Yet principled comparison is difficult unless there is a common language for describing what each module is, where it is inserted, what part of the host it interacts with, and how its contribution is integrated into the host computation.

This chapter addresses that problem through *PEFT-Ref*, a reference architecture and typology for describing PEFT techniques as reusable modules. PEFT-Ref provides a common descriptive vocabulary for the structural differences among the methods considered here. It complements the mathematical definition of each method by organising insertion locations, workspaces, integration forms, and related structural properties. These properties support comparison and hypothesis formulation; they do not, on their own, predict task performance.

Within the broader thesis, this chapter therefore provides the structural foundation of Part II. Later chapters ask functional questions about PEFT modules: whether they remain useful when transferred across host models (Chapter 5) and how limited adaptation resources can be organised under explicit budgets (Chapter 6). Those later questions build on the structural vocabulary established here.

Chapter outline. Section 4.1 presents the PEFT-Ref reference architecture. Section 4.2 defines the structural properties used to characterise PEFT modules. Section 4.2.1 positions PEFT-Ref relative to prior unifying work on PEFT. Section 4.3 applies the framework to seven widely used PEFT techniques. Section 4.4 compares those techniques with respect to efficiency and representative task-performance evidence. Section 4.5 derives evidence-grounded design implications and future module hypotheses. Section 4.6 summarises the chapter and links it forward to Chapters 5 and 6.

Notation. Throughout this chapter, d_m denotes the host-model hidden size. Where applicable, d_h denotes the hidden size of a PEFT module, or the bottleneck size in bottleneck-style adapter modules such as Houlsby-style Adapters. r denotes LoRA rank. n denotes prompt length for Prompt-Tuning or prefix length for Prefix-Tuning, depending on the technique under discussion. T denotes the input sequence length when analysing attention-related complexity.

4.1 Modular PEFT reference architecture

This section introduces the PEFT-Ref reference architecture used throughout the chapter. Its purpose is to provide a common structural map for describing where PEFT modules attach to a frozen host model, what workspace they act over, and how their outputs are integrated into host computation.

Figure 4.1 presents the PEFT-Ref reference architecture in diagrammatic form. Its role is not to suggest that all PEFT methods are used simultaneously, but to

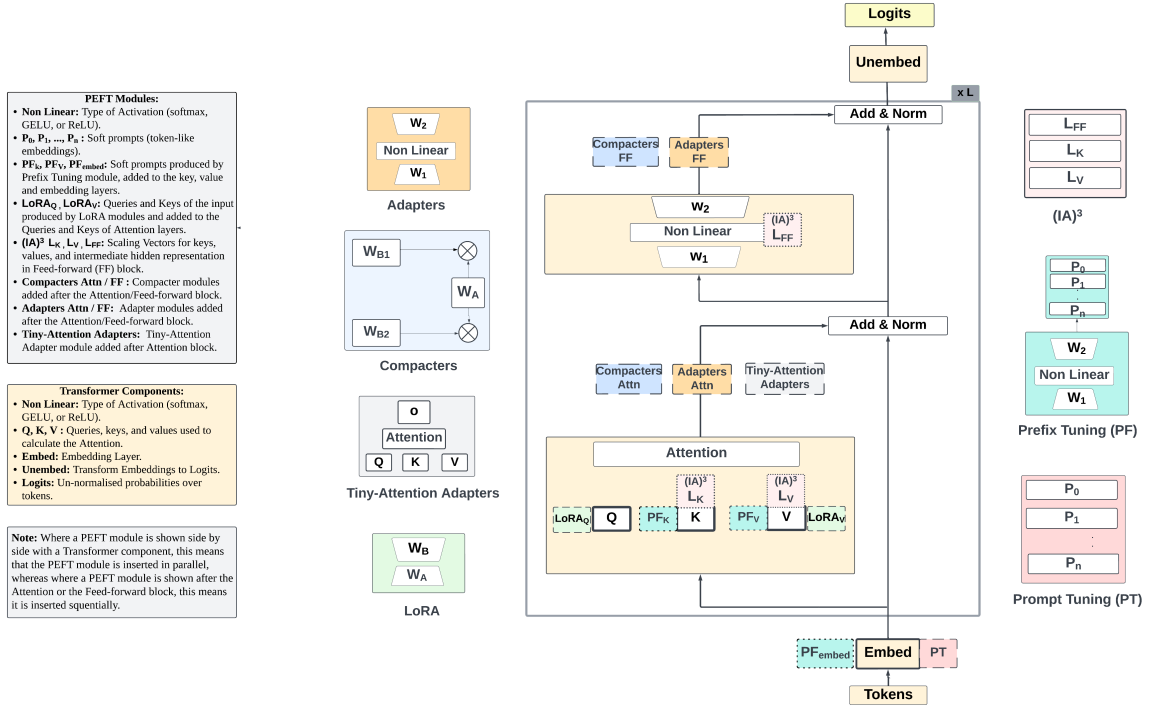


Figure 4.1: PEFT-Ref reference architecture. The central grey box shows the repeated Transformer layers and residual flow of the host model. Dashed regions indicate insertion slots used by different PEFT techniques. The diagram is intended as a structural reference: it shows where PEFT modules interface with the host model, what parts of the computation they interact with, and how different techniques differ in insertion location and workspace.

provide a common structural map against which different techniques can be described and compared. The repeated Transformer layers are shown inside the grey box, with residual flow to the right. Standard host-model components are shown in solid boxes, while dashed boxes indicate representative insertion slots used by PEFT modules.

The reference architecture serves a methodological purpose: it separates the host model from the inserted module and marks the locations at which PEFT techniques interface with host computation. It is therefore a descriptive aid for comparing methods in a shared vocabulary, rather than evidence that any one structural profile will perform better.

4.2 Structural properties of PEFT modules

PEFT-Ref instantiates, for the specific case of PEFT, the structural modularity perspective developed more generally in Chapter 2. The properties below are therefore not meant as a generic theory of modularity, but as a PEFT-specific set of structural variables that describe how a reusable PEFT module is built, where it interfaces with the host, and how it collaborates with host computation. Table 4.1 summarises the values of these properties for seven leading PEFT techniques.

The nine properties were selected using three practical criteria. First, taken together they must distinguish the module boundary, the host-module interface, the workspace acted on, and the form of integration. Second, each property must separate at least two of the surveyed methods without merely restating another property. Third, its values must be recoverable from the method definition and connect to an interpretable implementation or deployment consideration, such as insertion coverage, mergeability, parameter sharing, or added computation. The set is therefore a motivated descriptive basis for the seven methods analysed here, not an exhaustive or uniquely correct taxonomy of all PEFT designs.

1. **Intra-connectivity** (Clune et al., 2013; Meunier et al., 2010) — *dense*, *sparse*, or *none*. This describes the internal topology of the PEFT module itself, i.e., how the module’s own parameters or subcomponents are connected before its output is passed to the host model. A *dense* intra-connectivity means that the module contains a broadly connected computation block, such as an embedding layer, a multilayer perceptron, or a self-attention block. A *sparse* intra-connectivity would mean that only a restricted subset of internal connections is active within the module. A value of *none* means that the module does not contain an internally connected computation block at all, but instead consists only of directly learned parameter vectors or tensors. In this chapter, intra-connectivity matters because it partly determines module compute, parameter footprint, and the kind of transformation the module can perform.

-
2. **Inter-connectivity** (Clune et al., 2013; Meunier et al., 2010) — *fixed and dense*, *fixed and sparse*, or *dynamic*. This describes how the module’s output is connected to the host computation. A *fixed* connection means that the output-to-host interaction pattern does not depend on the input. Within this case, *fixed and dense* means that all output channels interact with the host, whereas *fixed and sparse* means that only a restricted subset of output channels does. A *dynamic* connection means that the effective output-to-host interaction pattern depends on the input, as in attention-based modules where the realised coupling is determined by input-conditioned attention weights rather than by a fixed MLP-style transformation. This property matters because it determines whether the module’s influence on the host is input-invariant or input-conditioned, and whether that influence is broadly distributed or restricted to a subset of output channels.

 3. **Parameters adapted** (Ding et al., 2023) — *direct addition* or *reparameterisation*. This describes how new trainable parameters are introduced into the adaptation. In *direct addition*, the PEFT method adds trainable parameters as an explicit module alongside the frozen host computation, without constraining them through a parameterisation of an existing host component. In *reparameterisation*, new trainable parameters are still introduced, but they are organised through a structured parameterisation of an existing host computation, for example by expressing an update in a constrained low-rank form. This distinction matters because it affects mergeability, parameter footprint, and the degree to which the PEFT method behaves as a separate additive module or as a structured reshaping of host computation.

 4. **Parameter sharing or tying** — *shared*, *tied*, or *none*. This describes whether different instances or subcomponents of a PEFT module reuse the same parameters or are constrained to remain similar. *Shared* means that the same parameters are reused across multiple parts of the module; *tied* means that distinct parameter sets are retained but encouraged to stay close; and *none*

means that no such relation is imposed. This property matters because sharing or tying can reduce parameter count, act as a regulariser, and introduce useful inductive bias (Ruder, 2017; Lan et al., 2020).

5. **Input type** (Pfeiffer et al., 2023; Auda and Kamel, 1999) — *hidden representations, data, or weights*. This describes the kind of object the PEFT module takes as its immediate input before producing its contribution to the host model. An input type of *hidden representations* means that the module operates on already contextualised activations from the host computation. An input type of *data* means that the module receives the same input signal as the host component it collaborates with. An input type of *weights* means that the module begins from directly learned parameter objects, such as prompt-like vectors or matrices, rather than from activations passed to it by the host. This property matters because it distinguishes modules that transform host-produced states from modules that alter conditioning or directly introduce learned parameter objects into the computation.
6. **Insertion form** (Pfeiffer et al., 2023; Auda and Kamel, 1999) — *sequential or parallel*. This describes how the PEFT module is positioned relative to the host computation it collaborates with. A *sequential* insertion means that the module receives the output of a host component and then feeds its own contribution back into the host flow. A *parallel* insertion means that the module receives the same input as the host component it collaborates with and contributes alongside it. This property matters because it affects both how the module interacts with host computation and what kinds of training, deployment, or mergeability trade-offs arise.
7. **Number of insertions** — *one layer, selected layers, or all layers*. This describes how widely instances of the PEFT module are distributed across the host model. A value of *one layer* means that the module is inserted at a single location, such as the embedding layer. A value of *selected layers* means

that module instances are inserted only at a chosen subset of host layers. A value of *all layers* means that the module is inserted throughout the repeated Transformer stack. This property matters because it affects parameter count, compute overhead, and the extent to which adaptation is localised to a small part of the host or distributed across the model more broadly.

8. **Integration form** (Auda and Kamel, 1999) — *concatenation, scaled addition, direct addition, gated addition, or rescaling*. This describes how the module’s output is incorporated into the host computation. *Concatenation* means that the module contributes additional representations that are appended to the host input at the relevant workspace. *Direct addition* means that the module output is added directly to the host computation. *Scaled addition* means that the added contribution is modulated by an explicit scaling factor. *Gated addition* means that the module contribution is controlled by an input- or parameter-dependent gate. *Rescaling* means that the module modulates existing host activations multiplicatively rather than by adding a new signal. This property matters because it affects how strongly the module can influence host computation, how explicitly that influence can be controlled, and whether the resulting contribution is easy to merge, suppress, or reinterpret at deployment time.
9. **Workspace** — *embedding layer, attention layer, feed-forward layer*, or a more specific locus within those. This property describes the part of the host model over which the PEFT module operates. An embedding-layer workspace means that the module acts at the level of input representations before the repeated Transformer computation begins. An attention-layer workspace means that the module acts on attention-related computations, such as projections or attention-conditioned representations. A feed-forward-layer workspace means that the module acts on the transformation of hidden states within the host model’s feed-forward subcomponents. More specific workspace values identify either a narrower locus within these broader regions, such as attention queries

and values or feed-forward intermediate representations, or a combination of loci, for example the embedding layer together with attention keys and values. This property is especially important in PEFT-Ref because it provides the clearest structural link between a module and the part of host computation it can most directly influence.

These PEFT-Ref properties can be related back to the broader structural properties introduced in Chapter 2. In particular, *boundary* concerns what counts as the PEFT module as a reusable object, and is captured here most directly through intra-connectivity, parameters adapted, parameter sharing or tying, and number of insertions. *Interface* concerns the signals exchanged between the module and the host model, and is refined here into input type, insertion form, and inter-connectivity. *Workspace* is retained as a separate property, since it identifies the part of the host system over which the module operates, rather than the signals or coupling pattern alone. *Integration form* is likewise retained directly, since it specifies how the module’s contribution enters the host computation.

Taken together, these properties are structural descriptors. Later in Part II I use them to formulate, rather than establish, hypotheses about functional reuse: Chapter 5 asks how structural differences are associated with portability, while Chapter 6 asks how selected properties can be controlled under explicit adaptation budgets.

4.2.1 Positioning relative to prior work

Chapter 2 reviews the broader literature on modularity and PEFT. Here I position PEFT-Ref relative to the most closely related attempts to unify and compare PEFT methods.

He et al. (2022) propose a unified functional view of several PEFT methods, describing common insertion patterns and composition operations. PEFT-Ref differs in two main ways. First, it provides a *reference architecture* that explicitly separates **where** a module interacts with the host model (its insertion location and workspace) from **how** its contribution is integrated. This avoids forcing all PEFT methods into

Technique	Intra-connectivity	Inter-connectivity	Parameters adapted	Sharing or tying	Input type	Insertion form	# tions	Inter-IntegrationWorkspace form
Prompt-Tuning	dense: embedding	fixed and dense	direct addition	none	weights	parallel	one layer	concatenation embedding layer
Prefix-Tuning	dense: non-linear MLP	fixed and dense	direct addition	none	weights	parallel	all layers + gated embedding layer	addi-embedding layer; attention keys and values
LoRA	dense: linear MLP	fixed and dense	reparameterisation	none	data	parallel	all layers	scaled addi-attention queries and values
Houlsby-style Adapters	dense: non-linear MLP	fixed and dense	direct addition	none	hidden representations	sequential	all layers	direct addi-attention and feed-forward blocks
Tiny-Attention Adapters	dense: self-attention	dynamic	direct addition	none	hidden representations	sequential	all layers	direct addi-attention block
Compacters	dense: non-linear MLP	fixed and dense	direct addition	shared	hidden representations	sequential	all layers	direct addi-attention and feed-forward blocks
(IA) ³	none	fixed and dense	direct addition	none	weights	sequential	all layers	rescaling attention keys and values; feed-forward intermediate representations

Table 4.1: PEFT-Ref structural characterisation of seven widely used PEFT techniques. The table summarises the property values defined in Section 4.2 and serves as the reference point for the technique-level discussion in Section 4.3.

a single down–nonlinearity–up template and accommodates prompt-like methods, rescaling methods, and attention-based variants. Second, PEFT-Ref defines a *typology of structural properties* with concrete values that support systematic side-by-side characterisation across a wider range of techniques.

Pfeiffer et al. (2023) present a broad unified view of modular deep learning, covering dimensions such as implementation and aggregation. PEFT-Ref complements this perspective by providing a more fine-grained and PEFT-specific description of module implementation: it makes explicit the insertion interface, workspace locus, and integration form in a way that directly supports later comparison and reuse analysis.

Survey and tooling efforts such as Ding et al. (2023) and UniPELT (Mao et al., 2022) contribute valuable empirical coverage and engineering support for training and composing PEFT methods. PEFT-Ref targets a different gap: a standardised descriptive scaffold that makes structural differences explicit and thereby supports principled comparison, interpretation of trade-offs, and later hypothesis generation about reuse behaviour.

4.3 Characterising PEFT techniques with PEFT-Ref

In this section, I apply PEFT-Ref to seven widely used PEFT techniques. I focus on methods that introduce and train topologically distinct adaptation components that can be treated as reusable PEFT modules. This scope excludes heuristic subset-finetuning approaches that update only selected host-model parameters (Lee et al., 2019; Zaken et al., 2022). The selected techniques are effective and widely used in downstream and practical settings (Ding et al., 2023; Liu et al., 2022). The seven descriptions show how each method instantiates a different reusable-module profile. Table 4.1 provides the reference summary; the technique-level discussion returns selectively to the relevant property values.

How PEFT-Ref is applied in this section. Each subsection begins with a compact PEFT-Ref profile stating the relevant property values. I then explain the technique in terms of those properties: what component is introduced, where it is inserted, what workspace it acts over, and how it collaborates with the host model. This is the main difference PEFT-Ref is meant to make in the chapter: it turns technique descriptions into explicit structural comparisons.

4.3.1 Prompt-Tuning

PEFT-Ref profile. Intra-connectivity = dense: embedding; Inter-connectivity = fixed and dense; Parameters adapted = direct addition; Sharing or tying = none; Input type = weights; Insertion form = parallel; # Insertions = one layer; Integration form = concatenation; Workspace = embedding layer.

Prompt-Tuning (Lester et al., 2021) introduces learned prompt vectors at the input-embedding level. For a learned prompt matrix $P \in \mathbb{R}^{n \times d_m}$ and token-embedding sequence $E(x)$, the frozen host receives the concatenated sequence $[P; E(x)]$; only P is updated. Through the PEFT-Ref lens, its defining structural choice is that the module lives entirely in the embedding workspace and is inserted only once, before the repeated Transformer layers. The module itself is a learned embedding table rather than a deeper transformation network.

This makes Prompt-Tuning computationally light and easy to store, but also means that the module influences the host model only through input conditioning. In PEFT-Ref terms, the module has a narrow workspace, one insertion point, and a fixed and dense interface with the host. Those properties help explain both its efficiency and the fact that it relies heavily on the frozen host model to propagate task-relevant information through later layers.

4.3.2 Prefix-Tuning

PEFT-Ref profile. Intra-connectivity = dense: non-linear MLP; Inter-connectivity = fixed and dense; Parameters adapted = direct addition; Sharing or tying = none;

Input type = weights; Insertion form = parallel; # Insertions = all layers; Integration form = gated addition; Workspace = embedding layer; attention keys and values.

Prefix-Tuning (Li and Liang, 2021) extends the prompt-based idea by learning a module g_ϕ that produces layer-specific prefix keys and values (K_ℓ^p, V_ℓ^p) . At layer ℓ , attention is evaluated as $\text{Attn}(Q_\ell, [K_\ell^p; K_\ell], [V_\ell^p; V_\ell])$, while the host parameters remain frozen. In PEFT-Ref terms, Prefix-Tuning differs from Prompt-Tuning along several dimensions at once: it uses a non-linear module internally, operates over an attention-related workspace, and is distributed across all layers rather than concentrated at the embedding layer alone.

4.3.3 LoRA

PEFT-Ref profile. Intra-connectivity = dense: linear MLP; Inter-connectivity = fixed and dense; Parameters adapted = reparameterisation; Sharing or tying = none; Input type = data; Insertion form = parallel; # Insertions = all layers; Integration form = scaled addition; Workspace = attention queries and values.

LoRA (Hu et al., 2021) reparameterises a selected frozen weight W through low-rank factors $A \in \mathbb{R}^{r \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r}$. Its forward map is $W_{\text{eff}}x = Wx + (\alpha/r)BAx$, with A and B trainable and W frozen. This is the clearest example in the chapter of the *reparameterisation* value of the ‘Parameters adapted’ property. Its module operates in parallel with the host computation and contributes through scaled addition.

PEFT-Ref makes the deployment-relevant features of LoRA explicit: it modifies selected attention projections, uses a low-rank reparameterisation, and can often be merged into the host weights at inference time. These properties are consistent with its low storage and serving overhead, but they do not by themselves explain its popularity or task performance. They matter again in Chapter 6, where the low-rank contribution is treated as one pathway in a budgeted adaptation setup.

4.3.4 Houlsby-style Adapters

PEFT-Ref profile. Intra-connectivity = dense: non-linear MLP; Inter-connectivity = fixed and dense; Parameters adapted = direct addition; Sharing or tying = none; Input type = hidden representations; Insertion form = sequential; # Insertions = all layers; Integration form = direct addition; Workspace = attention and feed-forward blocks.

Houlsby-style Adapters (Houlsby et al., 2019) insert bottleneck feed-forward modules after host subcomponents. For a hidden state h , a representative adapter computes $h' = h + W_{\text{up}}\sigma(W_{\text{down}}h)$, with the host frozen and the two adapter matrices trainable. Through the PEFT-Ref lens, their defining structural profile is a sequential insertion form operating on hidden representations rather than directly on weights or prompt-like inputs.

This structure makes Houlsby-style Adapters broad-purpose and easy to apply across layers, but it also means that they add extra hidden-state computation at inference time unless special compression or dropping schemes are used. PEFT-Ref also makes clear why adapter-family methods are easy to compare against each other: many of their variants keep the same workspace and integration form while altering other properties such as intra-connectivity or sharing.

4.3.5 Tiny-Attention Adapters

PEFT-Ref profile. Intra-connectivity = dense: self-attention; Inter-connectivity = dynamic; Parameters adapted = direct addition; Sharing or tying = none; Input type = hidden representations; Insertion form = sequential; # Insertions = all layers; Integration form = direct addition; Workspace = attention block.

Tiny-Attention Adapters (Zhao et al., 2022) replace the bottleneck feed-forward transformation with a small trainable attention module, giving a representative residual form $h' = h + \text{Attn}_{\phi}(h)$. The relevant PEFT-Ref distinctions are the change in intra-connectivity and the input-conditioned inter-connectivity of the added attention computation.

In the typology, the method preserves the adapter-style sequential integration pattern while changing the module’s internal topology and interaction dynamics. This is a descriptive distinction that can be tested against later reuse results.

4.3.6 Compacters

PEFT-Ref profile. Intra-connectivity = dense: non-linear MLP; Inter-connectivity = fixed and dense; Parameters adapted = direct addition; Sharing or tying = shared; Input type = hidden representations; Insertion form = sequential; # Insertions = all layers; Integration form = direct addition; Workspace = attention and feed-forward blocks.

Compacters (Karimi Mahabadi et al., 2021) preserve the adapter residual form but replace the dense adapter matrices with sums of Kronecker-product factors, for example $W = \sum_{i=1}^k A_i \otimes B_i$, with selected factors shared across layers. In PEFT-Ref terms, the defining difference from Houlsby-style Adapters lies primarily in the ‘Sharing or tying’ property rather than in workspace or insertion form.

The classification therefore places Compacters in the adapter family while recording their distinct parameter-sharing regime; the resulting parameter savings follow from the structured factorisation rather than from the classification itself.

4.3.7 (IA)³

PEFT-Ref profile. Intra-connectivity = none; Inter-connectivity = fixed and dense; Parameters adapted = direct addition; Sharing or tying = none; Input type = weights; Insertion form = sequential; # Insertions = all layers; Integration form = rescaling; Workspace = attention keys and values and feed-forward intermediate representations.

(IA)³ (Liu et al., 2022) does not introduce an internally connected module network. Instead, learned vectors rescale selected activations elementwise, for example $K' = l_k \odot K$, $V' = l_v \odot V$, and $f' = l_f \odot f$ for attention keys, values, and feed-forward intermediate activations. The host weights remain frozen. The key distin-

guishing properties are therefore `Intra-connectivity = none` and `Integration form = rescaling`.

Those properties make (IA)³ particularly lightweight at inference time and parameter-efficient in storage. They also distinguish it from the adapter family: although it is inserted throughout the host model, it collaborates with host computation by multiplicative rescaling instead of direct or scaled addition.

4.4 Efficiency and performance comparisons with PEFT-Ref

In this section, I use PEFT-Ref to compare the seven techniques along two broad dimensions: efficiency trade-offs and representative task-performance evidence. This focused comparison relates recurrent patterns to structural properties while retaining the reported benchmark outcomes and their limitations.

4.4.1 Efficiency trade-offs under the PEFT-Ref lens

Module compute and parameter footprint

Table 4.2 summarises each technique by (i) the time complexity per token of the added module computation and (ii) the number of parameters added per Transformer layer. Through the PEFT-Ref lens, the main structural drivers of these costs are **intra-connectivity**, **input type**, **workspace**, and **number of insertions**. In this comparison, I account only for the added module-side cost of producing the output that is then integrated into the frozen host model, not for the cost of the frozen host model itself.

Table 4.2 supports two descriptive comparisons. Techniques with minimal **intra-connectivity**, a narrow **workspace**, or limited **number of insertions**, such as Prompt-Tuning and (IA)³, have low module-side parameter and compute costs. Techniques can nevertheless occupy similar cost ranges while differing in **parame-**

Technique	Module complexity	Parameters per Transformer layer
Prompt-Tuning	$\mathcal{O}(1)$	$n d_m$
Prefix-Tuning	$\mathcal{O}(d_h d_m)$	$n d_m + d_m^2 + 2d_h d_m$
LoRA	$\mathcal{O}(r d_m)$	$4r d_m$
Tiny-Attention Adapters	$\mathcal{O}(T)$	$4 d_m$
Houlsby-style Adapters	$\mathcal{O}(d_h d_m)$	$4d_h d_m$
Compacters	$\mathcal{O}\left(\frac{d_h d_m}{N}\right)$	$4(d_h + d_m)$
(IA) ³	$\mathcal{O}(1)$	$6 d_m$

Table 4.2: Representative module-compute and parameter-footprint comparison for seven PEFT techniques. Here d_m is the model dimension; d_h is the module or bottleneck dimension where relevant; n is prompt or prefix length; r is LoRA rank; T is the input sequence length; and N is the reduction dimension in Compacters. Here, $\mathcal{O}(1)$ denotes modules whose output can be produced in a single step from their parameters, for example through a fixed-length embedding lookup or directly learned parameter vectors.

ters adapted, insertion form, integration form, and workspace; raw parameter count therefore does not specify the whole deployment profile.

Training dynamics: convergence and stability

Parameter efficiency does not necessarily translate into learning efficiency. Ding et al. (2023) report that full fine-tuning typically converges fastest, followed by Houlsby-style Adapters and LoRA, then Prefix-Tuning, while Prompt-Tuning converges slowest; as host models grow in size, PEFT convergence tends to improve. These findings suggest that convergence can be more sensitive to structural profile than to raw parameter count alone.

PEFT-Ref helps interpret those differences by making the relevant structural contrasts explicit. Prompt-Tuning combines **workspace = embedding layer** with **number of insertions = one layer**, so adaptation is localised to the input interface and must rely on the frozen host model to propagate task-specific effects through later layers. Relative to Prompt-Tuning, Prefix-Tuning broadens the **workspace** from the embedding layer alone to the embedding layer together with attention keys and values, and increases the **number of insertions** from one layer to all layers plus the embedding layer. This gives Prefix-Tuning deeper access to repeated attention computation, but it still depends on a module that produces prompt-like condition-

ing objects rather than directly modifying repeated hidden-state computation. By contrast, LoRA and Houlsby-style Adapters both interact with repeated layerwise computation throughout the Transformer stack, but do so through different PEFT-Ref profiles: LoRA uses **parameters adapted = reparameterisation**, **insertion form = parallel**, and **workspace = attention projections**, whereas Houlsby-style Adapters use **parameters adapted = direct addition**, **insertion form = sequential**, and **workspace = attention and feed-forward blocks**. These are not the only determinants of convergence, but they are structural differences that PEFT-Ref makes explicit and therefore interpretable.

Chen et al. (2022) additionally show that PEFT methods, like full fine-tuning, can exhibit performance fluctuations across seeds, and that reducing the number of trainable parameters can improve stability. Through the PEFT-Ref lens, this again points to a structural trade-off rather than a purely numerical one: choices about **parameter sharing or tying**, **number of insertions**, **input type**, and **integration form** can all affect optimisation behaviour.

Training-time gradient flow. In most PEFT settings, gradients are computed only for the module parameters while the host model remains frozen. The extent to which backpropagation must traverse the host computation graph depends most directly on **workspace**, **insertion form**, and **number of insertions**. Modules that operate earlier, such as prompt- or prefix-style conditioning at the embedding interface, require gradients to flow back to those earlier insertion points. Modules inserted locally within repeated Transformer blocks, such as LoRA or Houlsby-style Adapters, primarily require gradients through the local integration at those blocks. This is another example of a practical training consideration that PEFT-Ref makes easier to compare structurally.

Storage and deployment considerations

A major practical advantage of PEFT is storage: instead of saving a full fine-tuned model, one can store only the task-specific module parameters, often reducing

storage from gigabytes to megabytes. This enables one frozen host model to serve many tasks through multiple interchangeable PEFT modules. PEFT-Ref highlights which structural properties most directly control these savings, especially **number of insertions**, **workspace**, **intra-connectivity**, and whether the technique uses **direct addition** or **reparameterisation**.

Inference latency. Inference-time trade-offs also follow from PEFT-Ref properties. Prompt-Tuning and Prefix-Tuning primarily affect inference by consuming context length through their prompt-side conditioning, which reflects their **workspace** and **integration form**. LoRA is attractive in deployment partly because its **parameters adapted = reparameterisation** profile allows the low-rank update to be merged into the host weights (Hu et al., 2021), avoiding additional inference-time module computation. (IA)³ likewise introduces minimal latency because its **integration form = rescaling** is computationally light. In contrast, sequential hidden-state modules such as Houlsby-style Adapters and their variants combine **insertion form = sequential** with broader hidden-state **workspaces**, so they add explicit extra computation unless mitigated by methods such as AdapterDrop (Rücklé et al., 2021).

4.4.2 Representative task-performance evidence

This section summarises representative comparative evidence from the literature and relates the observed patterns back to PEFT-Ref properties. The goal is not to reproduce the full appendix collation in the main body, but to provide enough evidence here to support the design claims made later in the chapter. The fuller paper-by-paper table remains in Appendix B.1.1.

Table 4.3 reports that LoRA is frequently among the strongest methods in the selected comparative studies (Ding et al., 2023; Liu et al., 2022). It leads on sentiment analysis, emotion classification, multiple-choice question answering, and summarisation-style averages in those studies, while remaining competitive else-

Task family	Representative evidence	Leading techniques	PEFT-Ref interpretation
Sentiment classification	T5-base averages in Ding et al. (2023): LoRA is strongest on sentiment analysis (93.09) and emotion classification (68.70), ahead of Prefix-Tuning and Houlsby-style Adapters; Prompt-Tuning is clearly weaker.	LoRA strongest; Houlsby-style Adapters competitive; Prefix-Tuning slightly behind.	LoRA combines a lightweight structural profile with direct intervention in attention projections, giving it broad leverage without the deployment overhead of sequential hidden-state modules.
Natural language inference and hate-speech detection	In the same averages, Houlsby-style Adapters slightly outperform LoRA on natural language inference (83.06 vs. 82.73) and hate-speech detection (86.02 vs. 85.22), while both substantially outperform Prompt-Tuning (Ding et al., 2023).	Houlsby-style Adapters strongest; LoRA close behind.	Repeated sequential hidden-state adaptation over attention and feed-forward workspaces may be especially effective where broader layerwise adaptation matters.
Multiple-choice question answering and commonsense reasoning	For multiple-choice question answering, LoRA leads (58.67), ahead of Houlsby-style Adapters (56.11), Prefix-Tuning (53.93), and Prompt-Tuning (46.93) (Ding et al., 2023). In T0-3B few-shot commonsense reasoning, (IA) ³ is strongest (68.01), followed by LoRA (66.23) and Compacters (65.65) (Liu et al., 2022).	LoRA frequently strong; (IA)³ especially competitive in some reasoning-heavy settings.	The shared PEFT-Ref pattern is direct influence over repeated internal computation: LoRA via attention reparameterisation, and (IA) ³ via lightweight rescaling of attention- and feed-forward-related activations.
Summarisation and generation	In summarisation averages, LoRA (35.44) and Houlsby-style Adapters (35.38) are nearly tied and both outperform Prefix-Tuning (33.61) and Prompt-Tuning (30.35) (Ding et al., 2023). Other comparative studies also report Prefix-Tuning as competitive in generation settings (Li and Liang, 2021; Liu et al., 2022; Xu et al., 2022).	LoRA and Houlsby-style Adapters strong; Prefix-Tuning often competitive.	PEFT-Ref suggests a trade-off between richer conditioning breadth (Prefix-Tuning) and lighter parameter / deployment profiles (LoRA), while adapter-style methods provide broader sequential hidden-state interaction.
Word-sense disambiguation and fine-grained lexical tasks	On WiC, Compacters achieve the strongest score (55.33), narrowly ahead of LoRA (54.86), Houlsby-style Adapters (54.70), and (IA) ³ (54.23) (Liu et al., 2022).	Compacters strongest; LoRA and Houlsby-style Adapters competitive.	Here, PEFT-Ref highlights parameter sharing as the key differentiating structural property, suggesting that compact shared bottleneck-style modules can remain competitive on fine-grained discrimination tasks.

Table 4.3: Representative comparative findings brought into the main body from the literature surveyed in Appendix B.1.1. The fuller paper-by-paper collation remains in the appendix; the present table is intended to support the chapter’s qualitative comparative claims.

where. Its PEFT-Ref profile records attention-projection reparameterisation, parallel insertion, and scaled addition. These features describe how LoRA intervenes in repeated attention computation and why it can be lightweight at deployment; the comparisons do not isolate them as causes of task performance.

Adapter-family methods are also competitive in the selected studies and are strongest in some reported natural-language-inference and hate-speech comparisons (Ding et al., 2023; Liu et al., 2022). Their structural profile gives them repeated sequential access to hidden representations across attention and feed-forward workspaces. This breadth is a candidate explanation for the result, not a causal conclusion from the literature comparison.

No single PEFT-Ref profile dominates the task families included in Table 4.3 (Ding et al., 2023; Liu et al., 2022; Li and Liang, 2021; Xu et al., 2022). Prefix-Tuning is competitive in some generation settings, (IA)³ in some reasoning comparisons, and Compacters on WiC. The structural properties organise these comparisons and motivate hypotheses for later testing; they do not mechanically determine performance.

4.5 Implications of PEFT-Ref for module design and thesis progression

This section uses PEFT-Ref’s structural descriptions to formulate conditional hypotheses about module selection and future module design. The hypotheses connect explicit structural differences to the reuse, portability, and budgeted-organisation questions in later chapters. They require task-specific empirical testing and are not universal prescriptions.

4.5.1 Design implications for selecting PEFT modules

The observations below are intended as evidence-grounded design hypotheses rather than universal prescriptions. Their role in the thesis is to show how PEFT-Ref can

make technique choice more explainable by linking candidate task requirements to explicit structural properties.

Embedding-workspace conditioning under tight budgets. A first PEFT-Ref-based hypothesis is that techniques operating primarily through the *embedding workspace*, especially with low module complexity and limited insertion coverage, are low-cost candidates when the adaptation budget is tight and input-side conditioning is plausible. Prompt-Tuning has this profile: workspace = embedding layer, insertion form = parallel, and #insertions = one layer. Table 4.2 confirms its low module-side parameter and compute costs, while large-model results show that prompt-style conditioning can perform strongly on some tasks (Lester et al., 2021; Ding et al., 2023). These observations motivate including Prompt-Tuning in a budgeted comparison; they do not establish that it should be selected before alternatives without task-specific validation.

Attention workspaces for context-sensitive tasks. A second PEFT-Ref-based hypothesis concerns tasks that depend strongly on routing and combining contextual information. In the literature summarised in Table 4.3, LoRA is frequently competitive on question-answering-style tasks and other settings where attention management matters. Through the PEFT-Ref lens, the relevant properties are workspace = attention projections, parameters adapted = reparameterisation, and integration form = scaled addition. These properties provide a plausible structural explanation for why LoRA can work well in context-sensitive tasks: it intervenes directly in the computations that determine how information is selected and combined, while remaining relatively lightweight for deployment.

Generation tasks and the trade-off between conditioning breadth and efficiency. For generation tasks such as summarisation and data-to-text, both the evidence summarised in Table 4.3 and prior comparative studies suggest that Prefix-Tuning and LoRA can be strong options (Li and Liang, 2021; Liu et al.,

2022; Xu et al., 2022; Ding et al., 2023). PEFT-Ref helps articulate the trade-off. Prefix-Tuning broadens the workspace to include attention keys and values across layers, which may provide a richer conditioning signal. LoRA, by contrast, often achieves a more efficient deployment profile through low-rank reparameterisation and mergeability. This suggests a structural decision rule rather than a blanket prescription: when the budget is tighter, LoRA may be preferable; when richer prompt-side conditioning is worth the additional module cost, Prefix-Tuning may be attractive.

Feed-forward workspaces as a design hypothesis for knowledge-heavy tasks. PEFT-Ref also motivates a more speculative hypothesis: tasks with stronger dependence on stored world knowledge or controlled content realisation may benefit from modules that interact with feed-forward workspaces in addition to, or instead of, attention workspaces. This hypothesis draws on the broader literature linking Transformer feed-forward blocks to knowledge storage and retrieval (Dai et al., 2022) and on the PEFT-Ref observation that adapter-family methods naturally interact with hidden representations associated with attention and feed-forward computation. In the thesis, this hypothesis informed my reasoning about PEFT selection in data-to-text settings by making *workspace* an explicit design variable and avoiding the assumption that PEFT methods are interchangeable baselines. In the task context discussed in Mille et al. (2024), it directed attention to methods that can intervene in both attention-related and feed-forward computations associated with content storage and transformation. I use PEFT-Ref as a taxonomy for comparison and as a scaffold for task-specific module-selection hypotheses that can be tested directly.

4.5.2 Design hypotheses and future module directions

PEFT-Ref also makes it easier to state future module directions as explicit structural hypotheses rather than vague design intuitions.

Parameter sharing across module instances. The ‘Sharing or tying’ property is one such example. Parameter sharing and tying are well known to provide regularisation and inductive-bias benefits in broader neural-network design (Yeh et al., 2022). Within the PEFT techniques surveyed here, Compacters are the clearest example of this property. A natural extension is to explore sharing or tying within a module instance and across instances distributed over Transformer layers. The potential benefit would be to reduce parameter count while encouraging more stable or more general module behaviour. PEFT-Ref makes this extension explicit by recording sharing as a structural property. This hypothesis is not pursued empirically in later chapters, but is included here as a PEFT-Ref-motivated design direction.

Scaled addition for adapter-style modules. Another design direction concerns **Integration form**. LoRA uses scaled addition, whereas Houlsby-style Adapters use direct addition. A PEFT-Ref-motivated hypothesis is that introducing tunable scaling into adapter-style modules could improve controllability of module-host interaction, especially because such modules collaborate with multiple repeated host subcomponents across the Transformer stack. With direct addition, each adapter contribution is injected into the host flow without an explicit mechanism for controlling its magnitude relative to the frozen host computation. Scaled addition would instead provide a direct handle on how strongly the module perturbs the host at each insertion point. The potential benefit would be finer control over module influence and, potentially, greater stability when adaptation is distributed broadly across layers. This is not yet a standard design choice in vanilla Houlsby-style Adapters, though related work such as AdapterDrop shows that simple mechanisms for controlling adapter contribution or presence can already yield useful practical gains (Rücklé et al., 2021).

Selective insertion rather than uniform all-layer insertion. The ‘Number of insertions’ property also motivates future work. Most widely used PEFT techniques insert modules into all repeated Transformer layers (Ding et al., 2023; Pfeiffer et al.,

2023), but PEFT-Ref makes it easy to ask whether that should always be the default. A natural design hypothesis is that selective insertion into a subset of layers may preserve much of the useful adaptation signal while reducing parameter and inference cost. The benefit would be a better match between module placement and where adaptation is most needed. This idea is not yet dominant in standard PEFT practice, but heuristic fine-tuning work that selects only subsets of layers suggests a possible route for designing more selective PEFT modules (Lee et al., 2019).

Alternative workspaces beyond the standard set. Finally, the ‘Workspace’ property highlights a broader design space. By treating workspace as an explicit structural variable rather than leaving it implicit in the overall technique design, PEFT-Ref makes it natural to ask whether other reusable workspaces should be targeted. Most current PEFT methods operate over embedding, attention, or feed-forward workspaces (Ding et al., 2023; Pfeiffer et al., 2023). One possibility is the residual flow itself, i.e., the contextualised token representations that persist through the host network. Another is to design PEFT-style modules that collaborate with bias terms or LayerNorm-style components instead of directly fine-tuning them. The potential advantage would be to preserve the host model’s stored knowledge while still allowing targeted modulation of important computations. This direction is only partially explored in current work: heuristic fine-tuning methods such as BitFit and LayerNorm tuning (LN-Tuning) adapt these components directly (Zaken et al., 2022; Qi et al., 2022), but PEFT-Ref suggests a further question of whether explicit reusable modules could instead be built to interact with them.

4.6 Summary and thesis linkage

This chapter presented PEFT-Ref as a descriptive scaffold for treating PEFT techniques as reusable modules. It provides a structural vocabulary for comparing techniques in terms of *intra-connectivity*, *inter-connectivity*, *parameters adapted*, *parameter sharing or tying*, *input type*, *insertion form*, *number of insertions*, *integration*

form, and *workspace*. Its contribution in this thesis is organisational: explicit structural profiles supplement method names and benchmark results. The typology does not itself predict performance; it supplies variables and hypotheses for the empirical analyses that follow.

The next chapter takes the first functional step beyond structural description. If PEFT modules are to be treated as reusable artefacts in a functional sense, then structural characterisation alone is not enough: they must also retain utility when transferred outside their original training context. Chapter 5 therefore asks when PEFT modules are portable across hosts and settings. Chapter 6 then asks the next question: once reusable modules are identified, how can limited adaptation resources be organised across them under explicit constraints?

Chapter 5

PEFT-Port: On the Portability of Parameter-Efficient Fine-Tuning Modules

This chapter turns the structural account of PEFT modules in Chapter 4 into a functional question. Chapter 4 described PEFT techniques as modules with explicit interfaces, workspaces, and integration forms. Building on the conceptual framing in Chapter 2, the present chapter asks whether such modules retain useful task-specific behaviour when moved between paired checkpoints under constraints. The functional property studied here is *portability*.

Chapter outline. Section 5.1 motivates portability as a requirement for treating PEFT modules as reusable adaptation artefacts. Section 5.2 introduces the PEFT-Port protocol and comparison conditions, defines the chapter-local operational criteria, and positions the study relative to prior work on PEFT transfer and related reuse settings. Section 5.3 describes the experimental design, including host models, tasks, PEFT techniques, and matched-budget conventions. Section 5.4 presents a controlled-baseline study that tests distribution-matched initialisation as an alternative explanation for the observed porting advantage. Section 5.5 reports an extended portability study across tasks, models, and budgets, including zero-shot

non-collapse rates and matched-budget portability criteria. Section 5.6 discusses the dominant factors that govern portability and interprets trends through PEFT-Ref properties (Chapter 4). Section 5.7 connects the results to budgeted adaptation in Chapter 6 and to the broader thesis narrative. Finally, Section 5.8 provides a concise chapter summary.

5.1 Why portability matters for PEFT as reusable modules

In Chapter 4, I treated PEFT components as *PEFT modules*: detachable adaptation components attached to a frozen host model, intended to support reuse through storage, swapping, comparison, and budgeted adaptation. That structural account is necessary, but it is not sufficient. A module is not reusable in a functional sense merely because it is separable; it must also preserve useful behaviour when moved outside the exact conditions in which it was trained.

As discussed conceptually in Chapter 2, this thesis distinguishes between the structural characterisation of a module and reuse-relevant functional properties such as portability. Chapter 4 provided that structural characterisation for PEFT techniques by defining them as reusable PEFT modules with explicit interfaces, workspaces, and integration forms. The present chapter builds on that foundation for the external PEFT-module side of the thesis.

This chapter operationalises portability in a deliberately narrow setting: a module trained on one checkpoint is imported into the corresponding same-scale raw or instruction-tuned T5-family checkpoint and evaluated under matched post-porting budgets. The hosts share architecture, tokenizer, tensor shapes, and a training lineage; they are not independently initialised same-architecture models. Chapter 5 therefore tests whether useful behaviour survives this controlled checkpoint-to-checkpoint transfer, not whether arbitrary modules can move between independently trained hosts.

5.2 Preliminaries: portability protocol, controls, and operational criteria

This section establishes the conceptual and operational scaffolding needed to interpret the PEFT-Port experiments. Section 5.2.1 introduces the porting protocol and comparison conditions, Section 5.2.2 defines the operational criteria reported in the results, and Section 5.2.3 positions the study relative to prior work on PEFT transfer and related reuse settings.

5.2.1 PEFT-Port protocol and comparison conditions

As discussed in Chapter 3, reuse claims in this thesis are evaluated under controlled budgets and against baselines that rule out trivial explanations. PEFT-Port instantiates that general methodological principle for cross-host transfer of PEFT modules.

At a high level, the protocol has two stages. First, a PEFT module is trained on an exporting host model. Second, the learned parameters are imported into the corresponding same-scale host from the paired T5 v1.1/Flan-T5 checkpoint family, where the module is either evaluated immediately or allowed additional post-porting adaptation under a matched importing-side budget. Zero-shot portability is the special case in which no post-porting updates are allowed.

This general porting-then-adaptation structure requires an initialisation control. An imported module might help because its parameters lie in a statistically favourable region of the receiving model’s optimisation landscape, even when little task-specific knowledge transfers. I therefore test whether the observed gain reflects functional encapsulation in the ported parameters or can be reproduced by a distribution-matched random initialisation.

PEFT-Port evaluates portability with explicit controls. Across this chapter, I report two complementary experimental grids: (i) a controlled-baseline study that includes a distribution-matched initialisation control, and (ii) a broader full study

across tasks, models, and budgets.

To isolate genuine transfer, I evaluate the following scenarios:

- **Ported:** train a PEFT module on an exporting host model, then import the learned parameters into an importing host model.
- **From scratch:** train the same PEFT technique on the importing host model from its default random initialisation.
- **Distribution-matched (sampled):** initialise a fresh importing-side module by independently sampling each parameter tensor from a Gaussian distribution that matches the corresponding ported tensor’s empirical mean and variance, then train it under the same importing-side budget. The complete tensor-wise sampling procedure is specified in Appendix C.2.3.

The distribution-matched baseline is used only in the controlled-baseline study to test whether favourable first- and second-order parameter statistics can explain the observed porting advantage. Appendix C.2.3 provides the complete construction of this control.

Pre-porting and post-porting training. I refer to training on the exporting side as *pre-porting* training. After import, I optionally allow *post-porting* adaptation training under a controlled budget.

5.2.2 Operational definitions and reported measures

Building on the conceptual framing in Chapter 2 and the shared methodological conventions in Chapter 3, I operationalise the reported portability measures in this chapter as follows.

Zero-shot portability. *Zero-shot portability* refers to the case where the imported module is evaluated on the importing host with *no* post-porting parameter updates.

Zero-shot non-collapse. *Zero-shot non-collapse* refers to the case where the imported module, evaluated with no post-porting updates, achieves non-zero task performance on the importing host. I use this measure only as a weak indicator that importing does not completely destroy task behaviour. Because non-zero performance can still arise from the importing host or from task-specific chance behaviour, this criterion is weaker than the later matched-baseline portability tests and should not be interpreted on its own as sufficient evidence of useful transferred functionality.

Recovery of exporting-side performance. I say that a ported module *recovers exporting-side performance* when, after post-porting adaptation, its importing-side performance reaches at least 99% of the corresponding exporting-side performance obtained before porting. If P_{pre} denotes the exporting-side pre-porting score and P_{post} denotes the importing-side post-porting score, then recovery is counted when

$$\frac{P_{\text{post}}}{P_{\text{pre}}} \geq 0.99.$$

I use 99% as a stringent operational threshold that permits a one-percentage-point relative tolerance instead of requiring exact numerical equality. It is not a statistical-equivalence margin or a universal definition of successful portability. Counts based on it are conditional on this threshold and on the seed-averaged scores; nearby cases could change category under a different tolerance. Accordingly, the recovery and joint-criterion percentages are descriptive summaries of the evaluated grid, not binary evidence that a PEFT technique is intrinsically portable.

Outperforming from scratch. A ported module is counted as *outperforming from scratch* when, under the same post-porting learning-step budget, it achieves a higher score than an architecturally identical importing-side module trained from default initialisation.

Meeting both criteria. The strictest criterion used in the chapter requires that a ported module both (i) outperform the matched from-scratch baseline and (ii)

recover at least 99% of its exporting-side performance.

Cases and percentages. Unless otherwise stated, a *case* denotes one unique experimental configuration in the grid after aggregating over the corresponding repeated-seed runs: a particular combination of PEFT technique, task or dataset setting, host-model pair, porting direction, and pre-/post-porting budget. The percentages reported in the results tables are therefore proportions of such cases satisfying the relevant criterion.

5.2.3 Positioning relative to prior work

Chapter 2 covers the broader landscape of PEFT and modular deep learning. Here I position PEFT-Port relative to the most closely related prior work on PEFT transfer, and to adjacent reuse settings that become relevant when asking how PEFT modules are reused in practice.

One such related setting is *compositionality*: the question of whether separately trained PEFT modules can be combined productively within a shared host model. Prior work has explored this most notably for Houlsby-style Adapters in multilingual and multi-task settings (Pfeiffer et al., 2020). In those studies, the main interest is typically practical reuse within a single host, for example combining task and language modules, rather than testing cross-host portability. More broadly, other work has studied *transferability* without changing the host model, such as cross-task transfer in prompt tuning (Su et al., 2022; Vu et al., 2022) and broader examinations of task-level transfer across PEFT techniques (Ding et al., 2023). These studies show that PEFT modules can capture reusable behaviour, but they do not usually ask whether that behaviour survives movement across host models.

This difference in emphasis matters. Much of the prior literature is motivated by practical transfer-learning questions in deep learning, such as whether adaptation learned in one task or setting can be reused elsewhere, rather than by an explicit modularity-for-reuse framework. In this thesis, by contrast, I adopt that modular

perspective deliberately: building on Chapter 2 and Chapter 4, PEFT components are treated as reusable PEFT modules, and properties such as compositionality and portability are interpreted as reuse-relevant functional properties of those modules.

The present chapter targets **paired-checkpoint portability**. More specifically, PEFT-Port asks whether a PEFT module trained on one host can be imported into its corresponding same-scale raw or instruction-tuned checkpoint and remain useful under controlled post-porting budgets. The chapter also includes a distribution-matched initialisation control, so that an aggregate advantage of porting can be compared with a baseline that preserves tensor-wise means and variances but not the learned parameter arrangement.

Within this paired-checkpoint setting, a matched-budget advantage is evidence that the learned parameter arrangement retains useful task information beyond its exporting checkpoint. It is not evidence that architectural compatibility alone is sufficient for portability. Independently trained same-architecture models can implement features in unaligned parameter bases, so equal tensor shapes provide an interface but not necessarily a compatible internal coordinate system. Testing such hosts would require separate alignment methods or experiments and remains outside this chapter’s scope.

5.3 Experimental design and common scaffolding

As discussed in Chapter 3, reuse claims in this thesis are evaluated under controlled budgets and against baselines that rule out trivial explanations. The PEFT-Port study instantiates that shared methodological scaffold for cross-host transfer of PEFT modules. The porting protocol itself was introduced in Section 5.2.1; the purpose of the present section is therefore not to restate that protocol, but to specify the concrete experimental grid used to implement it in this chapter.

For the from-scratch condition, the importing-side module is initialised using the standard initialisation of the corresponding PEFT method on the importing host model. The full experimental grids and host-model pairings are reported in

Appendix C.2.1; PEFT-specific and optimisation settings are reported in Appendices C.2.5 and C.2.6, respectively.

Two complementary experimental grids. I report results from two PEFT-Port experimental grids. First, a **controlled-baseline study** comprising 1,440 training/testing runs, reported in Section 5.4, which adds the *distribution-matched (sampled)* initialisation control to distinguish genuine transfer from favourable initialisation. Second, a **full study** comprising 5,757 training/testing runs, reported in Section 5.5, which expands the grid across multiple tasks, datasets, host-model directions, and pre-/post-porting budgets to characterise portability patterns more broadly.

Appendix C.2.1 summarises the two experimental grids, including their host-model pairs, budgets, conditions, and seeds. Supplementary implementation details are organised by purpose: dataset and split policy in Appendix C.2.2, evaluation and aggregation in Appendix C.2.4, PEFT and training hyperparameters in Appendices C.2.5 and C.2.6, and computational resources in Appendix C.2.7.

5.3.1 Host models

I use open-source T5 v1.1 models and their corresponding same-scale instruction-tuned Flan-T5 variants.

This choice serves two purposes. First, it tests transfer across a practically relevant change in checkpoint state: before and after instruction tuning. Second, the paired checkpoints provide a controlled way to probe the role of *pre-existing task-relevant knowledge*: instruction tuning can change task usability and zero-shot performance (Chung et al., 2022), which may alter how much additional task information a PEFT module must encode.

I therefore evaluate both directions (raw $\leftrightarrow$ instruction-tuned) and analyse direction effects in Section 5.5.5. The corresponding checkpoints share architecture, tokenizer, layer shapes, and training lineage; the instruction-tuned host continues

Instruction-tuned host	Raw host	#Params
Flan-T5 Base	T5 v1.1 Base	250M
Flan-T5 Large	T5 v1.1 Large	780M
Flan-T5 XL	T5 v1.1 XL	3B

Table 5.1: Corresponding raw and instruction-tuned host models used in PEFT-Port. Porting is performed only between same-scale host pairs.

from a pretrained T5-family state rather than from an independent random initialisation. This alignment makes direct tensor import feasible, but also limits the inference: the study does not test different raw hosts, different instruction-tuned hosts, or independently trained models that merely share an architecture. Architectural compatibility is necessary for the direct import used here, but it is not sufficient to guarantee functional portability.

Table 5.1 summarises the host models used. Porting is restricted to corresponding same-scale raw and instruction-tuned hosts: Base $\leftrightarrow$ Base, Large $\leftrightarrow$ Large, and, in the full study, XL $\leftrightarrow$ XL. No cross-scale porting is evaluated. The complete host-pair and direction specification is provided in Appendix C.2.1.

5.3.2 Tasks and datasets

To keep evaluation comparable across raw and instruction-tuned hosts, tasks are formulated as sequence prediction: the input is provided directly as the prompt, without task descriptions or instruction prefixes, and the generated continuation is interpreted as the output label or text.¹

Table 5.2 summarises the task families, metrics, and datasets. Sentiment analysis, natural-language inference, and paraphrase detection were selected because each provides a paired same-task/different-dataset comparison (Rotten Tomatoes/SST-2, MNLI/SICK, and MRPC/QQP), separating host transfer from an additional dataset shift. SAMSum adds a generation task with a different metric and output struc-

¹Instruction-tuned models can benefit disproportionately from explicit task descriptions and instruction-style formatting (Chung et al., 2022; Wei et al., 2022). Using a minimal, prefix-free formulation avoids conflating portability with differences in instruction-following behaviour and keeps the evaluation protocol consistent across raw and instruction-tuned hosts.

Task family	Description	Metric	Datasets
Sentiment analysis	Identifying the sentiment polarity in a given sentence.	Accuracy	SST-2 (Socher et al., 2013); Rotten Tomatoes (Pang and Lee, 2005)
Natural language inference	Determining the relationship between a premise and hypothesis pair: whether the hypothesis is entailed by, contradicts, or is unrelated to the premise.	Accuracy	MNLI (Williams et al., 2018); SICK (Marelli et al., 2014)
Paraphrase detection	Determining whether a pair of sentences is semantically identical.	Accuracy	QQP (Sharma et al., 2019); MRPC (Dolan and Brockett, 2005)
Summarisation	Summarising a given article.	ROUGE-L	SAMSum (Gliwa et al., 2019)

Table 5.2: Task families and datasets used in the PEFT-Port study.

ture. The controlled-baseline study uses sentiment because its paired datasets and accuracy metric support a tractable, easily interpreted initialisation comparison; the full study then broadens the evaluation to all four task families and seven datasets. The exact prompt formats, target outputs, decoding and scoring procedures, and dataset-split policy are documented in Appendix C.2.4 and Appendix C.2.2, respectively.

5.3.3 PEFT techniques

I evaluate four PEFT techniques chosen to span distinct structural profiles in PEFT-Ref: Housby-style Adapters provide sequential hidden-representation modules; Prefix-Tuning provides context injection through attention keys and values; LoRA provides direct low-rank weight reparameterisation; and Compacter provides a parameter-shared, structured adapter. This selection enables a controlled comparison across

PEFT	Architecture	Repeats	Insertion	Workspace	Compatibility requirements
Prefix-Tuning	non-linear MLP	All layers	Parallel	Embedding; attention keys/values	Model dimension, #layers
LoRA	linear MLP	All layers	Parallel	Attention query/values	Model dimension, #layers
Houlsby-style Adapters	non-linear MLP	All layers	Sequential	FFN and attention block	Model dimension, #layers
Compacter	non-linear MLP	All layers	Sequential	FFN and attention block	Model dimension, #layers

Table 5.3: Structural properties of the PEFT techniques used in this chapter, expressed using the PEFT-Ref typology introduced in Chapter 4. The final column summarises the basic architectural compatibility required to import a module without changing its interface. *Note:* Compacter uses a Kronecker-product reparameterisation of the Houlsby-style Adapters MLP.

insertion form, workspace, and parameter structure; it is not intended to be exhaustive of PEFT. The exact method-specific hyperparameters are reported in Appendix C.2.5 (Houlsby et al., 2019; Li and Liang, 2021; Hu et al., 2021; Karimi Mahabadi et al., 2021).

All four insert modules across Transformer layers, but they differ in where they intervene. Houlsby-style Adapters and Compacter primarily modulate *hidden representations*, whereas Prefix-Tuning and LoRA intervene through *attention-side projections*. This distinction is relevant because portability may depend on how tightly a module’s parameters are coupled to host-specific parameter structure, for example attention projections, versus acting through more representation-level transformations. Their key PEFT-Ref structural properties are summarised in Table 5.3.

For readability in figures, I use *Adapters* as shorthand for *Houlsby-style Adapters*.

5.3.4 Budgets, steps, and seeds

The controlled-baseline study evaluates pre-porting budgets of 5K and 10K steps and evaluates imported modules immediately after porting and after 0.5K, 1.5K, and 3K post-porting steps across three random seeds: 124, 5090, and 675323. The full study expands the pre-porting budgets to 5K, 10K, 20K, and 40K steps and evaluates imported modules immediately after porting and after 0.5K, 1.5K, 3K, and 5K post-porting steps. It uses two random seeds, 124 and 5090, and the reported full-grid performance results are averaged across them. Appendix C.2.1 provides the complete study-level grid, while Appendix C.2.6 reports the optimisation settings, batch sizes, and checkpointing procedures used within those budgets.

The 5K and 10K pre-porting checkpoints sample early adaptation, while 20K and 40K sample more mature exporting-side modules; the post-porting checkpoints similarly cover immediate transfer and progressively larger importing-side budgets. These are discrete operating points chosen to reveal budget dependence, not a claim that each method is evaluated at its optimum. Three seeds were affordable for the narrower controlled study; the much larger full grid uses two seeds to keep the total experiment tractable. Consequently, the full-grid means and threshold counts have limited precision and should be read as descriptive patterns rather than fine-grained rankings when differences are small.

5.4 Controlled-baseline study: testing favourable initialisation

5.4.1 Study-specific experimental set-up

Before scaling to the full experimental grid, I first conducted a controlled-baseline study designed to answer a narrower question: can the apparent benefits of portability be explained primarily by favourable initialisation? This study comprises 1,440 training/testing runs and compares three importing-side conditions under matched

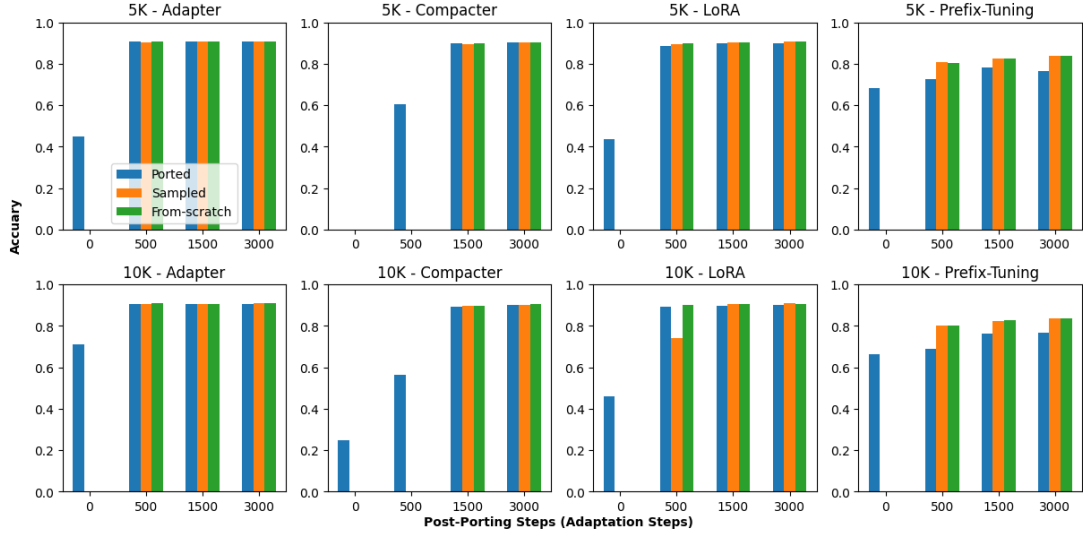
post-porting budgets: *ported*, *sampld* (distribution-matched initialisation), and *from scratch*. Unlike the broader full study in Section 5.5, this study is restricted to sentiment analysis and is intended specifically to test whether the main portability signal survives a stronger initialisation control. The exact distribution-matched sampling procedure is specified in Appendix C.2.3, and the controlled-study grid is summarised in Appendix C.2.1.

5.4.2 Aggregate results

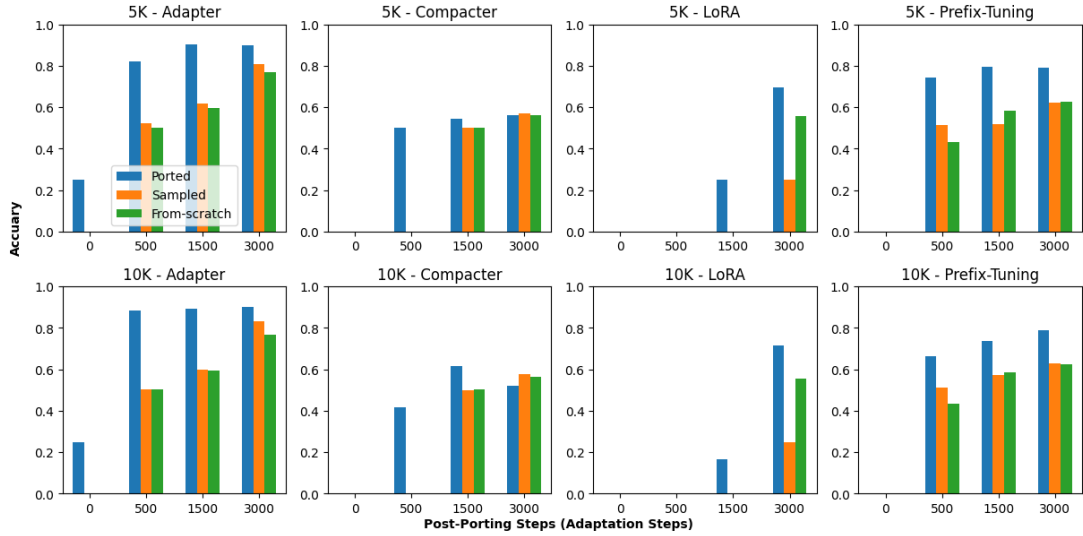
Rows in Table 5.4 separate same-dataset and cross-dataset settings; columns compare ported, distribution-matched sampled, and from-scratch conditions. Values are mean accuracy with variance, aggregated over host pairs, budgets, and seeds. In every PEFT row, the ported condition has the highest aggregate mean. For example, same-dataset Houlsby-style Adapters achieve 0.895 when ported, compared with 0.777 when sampled and 0.765 from scratch; the cross-dataset values are 0.930, 0.797, and 0.785. The sampled control therefore does not recover the aggregate porting advantage. This is consistent with information being carried by the learned parameter arrangement rather than by tensor-wise mean and variance alone, although the control does not exclude every possible initialisation-related explanation.

5.4.3 Porting-direction effects

Figure 5.1 separates the same- and cross-dataset settings by porting direction and post-porting budget. Ported modules are blue, distribution-matched modules orange, and from-scratch modules green. Porting advantages appear at small budgets in many panels, while raw→instruction-tuned transfer shows stronger zero-step and low-budget gains for several techniques than the reverse direction. The controlled study therefore provides two bounded findings: tensor-wise distribution matching does not reproduce the aggregate ported result, and the observed advantage is direction-dependent. Section 5.5.5 examines the latter pattern in the full study.

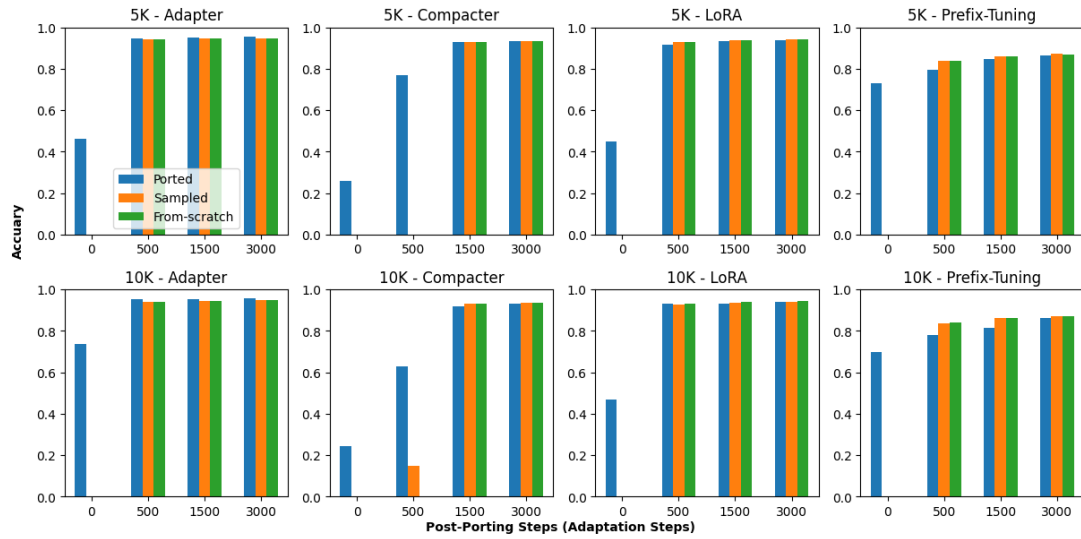


(a) Same dataset: raw $\rightarrow$ instruction-tuned.

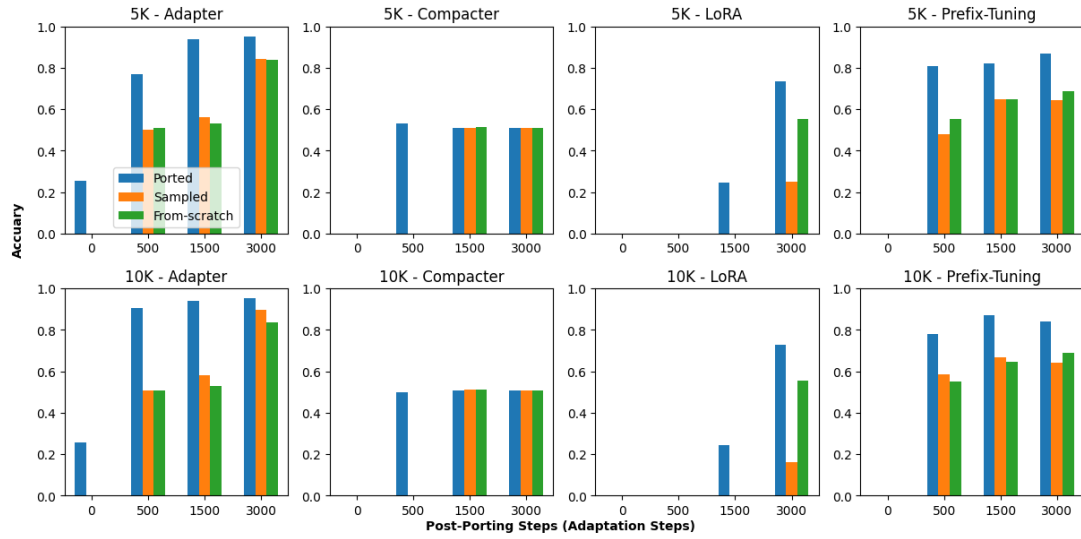


(b) Same dataset: instruction-tuned $\rightarrow$ raw.

Figure 5.1: Controlled-baseline results. Each chart shows average accuracy over three seeds and the Base/Large host pairs for one PEFT technique, direction, and pre-porting budget. Accuracy is plotted against post-porting steps; blue = ported, orange = distribution-matched sampled, and green = from scratch. The panels are enlarged and continued on the following page for readability.



(c) Different datasets: raw $\rightarrow$ instruction-tuned.



(d) Different datasets: instruction-tuned $\rightarrow$ raw.

Figure 5.1: Controlled-baseline results (continued): different-dataset transfer.

Setting	PEFT	Ported	Sampled	From scratch
Same dataset (RT $\rightarrow$ RT)	Houlsby-style Adapters	0.895 (0.001)	0.777 (0.031)	0.765 (0.033)
	Compacter	0.661 (0.037)	0.478 (0.140)	0.477 (0.140)
	LoRA	0.600 (0.148)	0.480 (0.189)	0.544 (0.166)
	Prefix-Tuning	0.751 (0.010)	0.692 (0.021)	0.685 (0.032)
Different datasets (RT $\rightarrow$ SST-2)	Houlsby-style Adapters	0.930 (0.005)	0.797 (0.040)	0.785 (0.041)
	Compacter	0.681 (0.037)	0.493 (0.147)	0.481 (0.147)
	LoRA	0.629 (0.157)	0.502 (0.205)	0.561 (0.179)
	Prefix-Tuning	0.829 (0.005)	0.734 (0.027)	0.743 (0.020)

Table 5.4: Mean accuracy (variance) for ported, sampled (distribution-matched), and from-scratch scenarios on sentiment analysis. Values are aggregated over exporting/importing model pairs (Base/Large; raw/Flan), pre-/post-porting step budgets, and random seeds. RT denotes Rotten Tomatoes.

5.5 Full portability study: transfer across tasks, models, and budgets

5.5.1 Study-specific experimental set-up

The full study extends the controlled-baseline analysis in Section 5.4 to a broader grid spanning multiple tasks, datasets, host-model pairs, porting directions, and pre-/post-porting budgets. Its purpose is not to revisit whether favourable initialisation can explain the porting signal; that question was addressed directly in Section 5.4. Instead, the full study asks how portability behaves once the analysis is scaled across task families, model scales, and adaptation budgets. Accordingly, I report results at three levels: (i) overall performance patterns across the full experimental grid, (ii) zero-shot non-collapse as a weak indicator that importing does not completely destroy task behaviour, and (iii) matched-budget portability criteria that compare imported modules with from-scratch importing-side modules and with their exporting-side performance.

The full-study grid, including the two seeds and same-scale host pairings, is reported in Appendix C.2.1; the seed-aggregation and case-counting procedures used for the reported percentages are specified in Appendix C.2.4.

5.5.2 Descriptive overview of the full experimental grid

The full study spans four task families (sentiment analysis, natural language inference, paraphrase detection, and summarisation), seven datasets, multiple host-model pairs, two porting directions (raw $\leftrightarrow$ instruction-tuned), and a range of pre- and post-porting budgets.

Figure 5.2 separates same-dataset and different-dataset scenarios. Rows give pre-porting budgets, columns give porting directions, coloured bars give the four imported PEFT methods across post-porting budgets, black diamonds give matched from-scratch baselines, and horizontal lines give exporting-side performance.

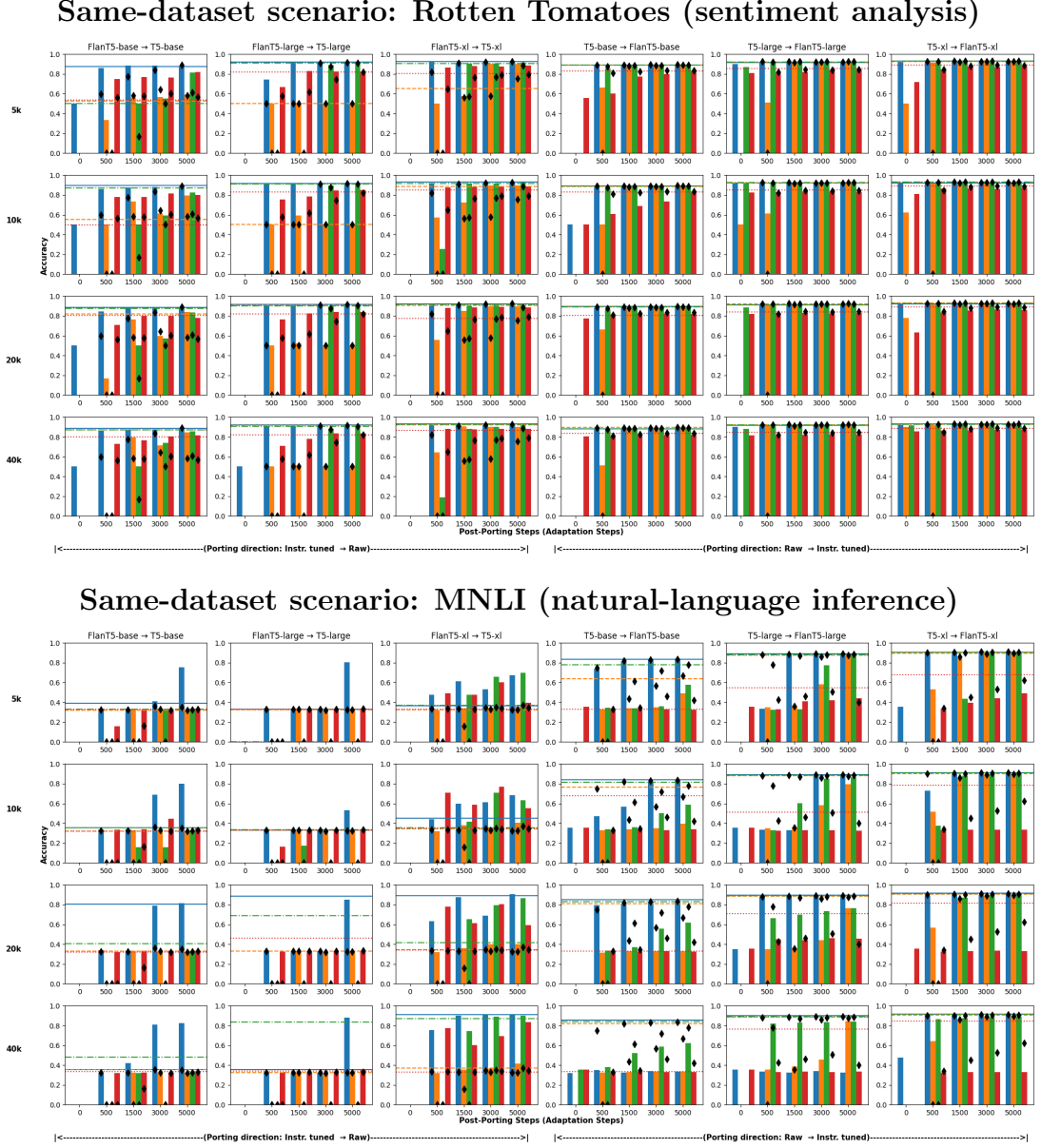


Figure 5.2: Performance of ported PEFT modules; black diamonds denote matched from-scratch modules. Results are averaged over two seeds. Within each task panel, rows are pre-porting budgets (5K, 10K, 20K, and 40K), columns are porting directions, and the x-axis gives post-porting steps (0, 0.5K, 1.5K, 3K, and 5K). The seven task panels are enlarged across four consecutive pages so that labels and values remain readable.

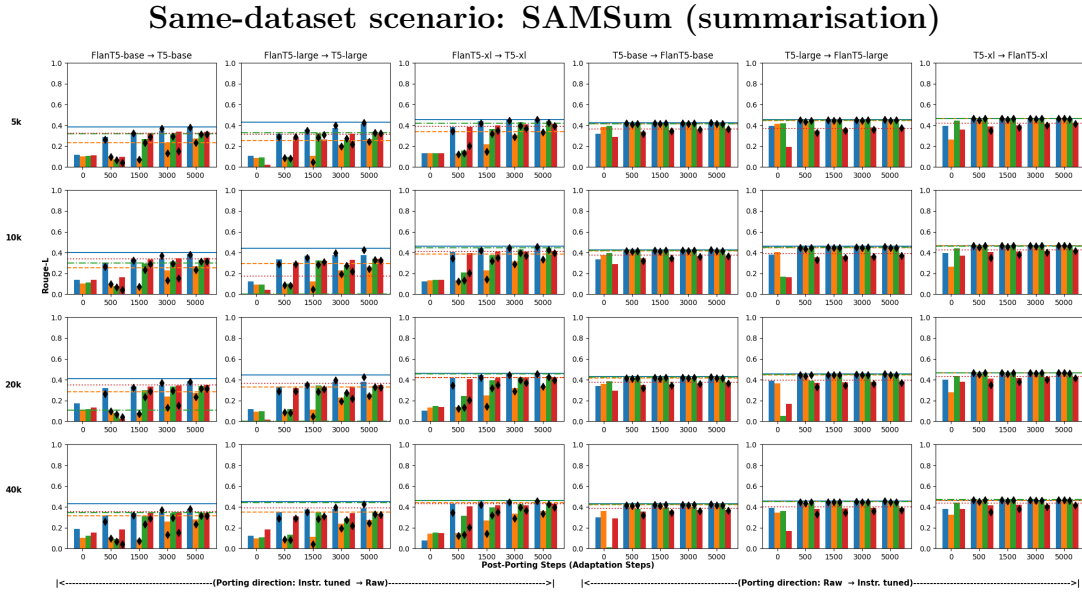
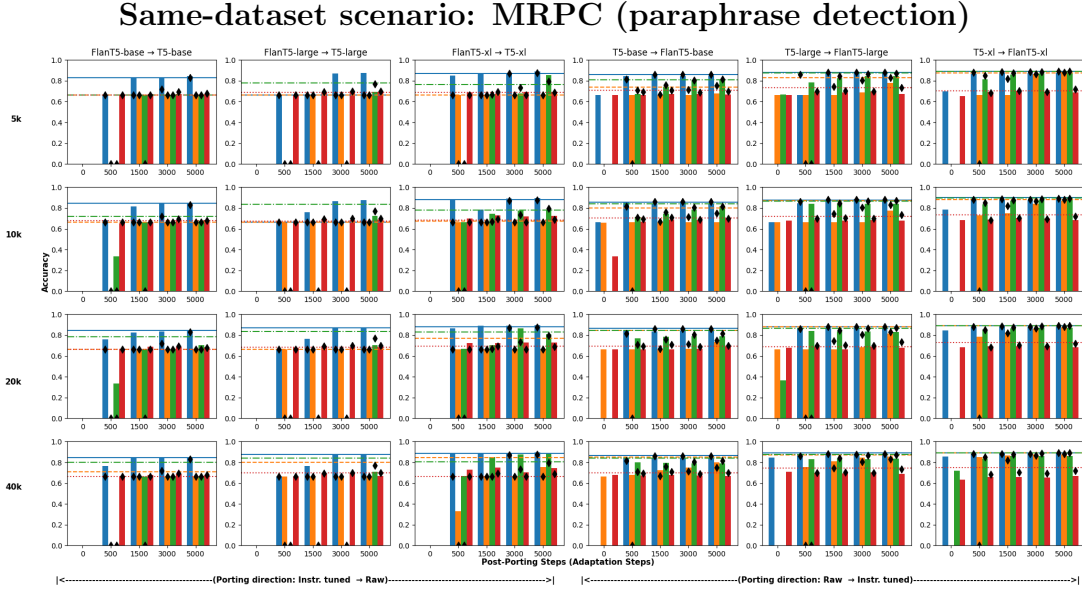


Figure 5.2: Performance of ported PEFT modules (continued): remaining same-dataset tasks.

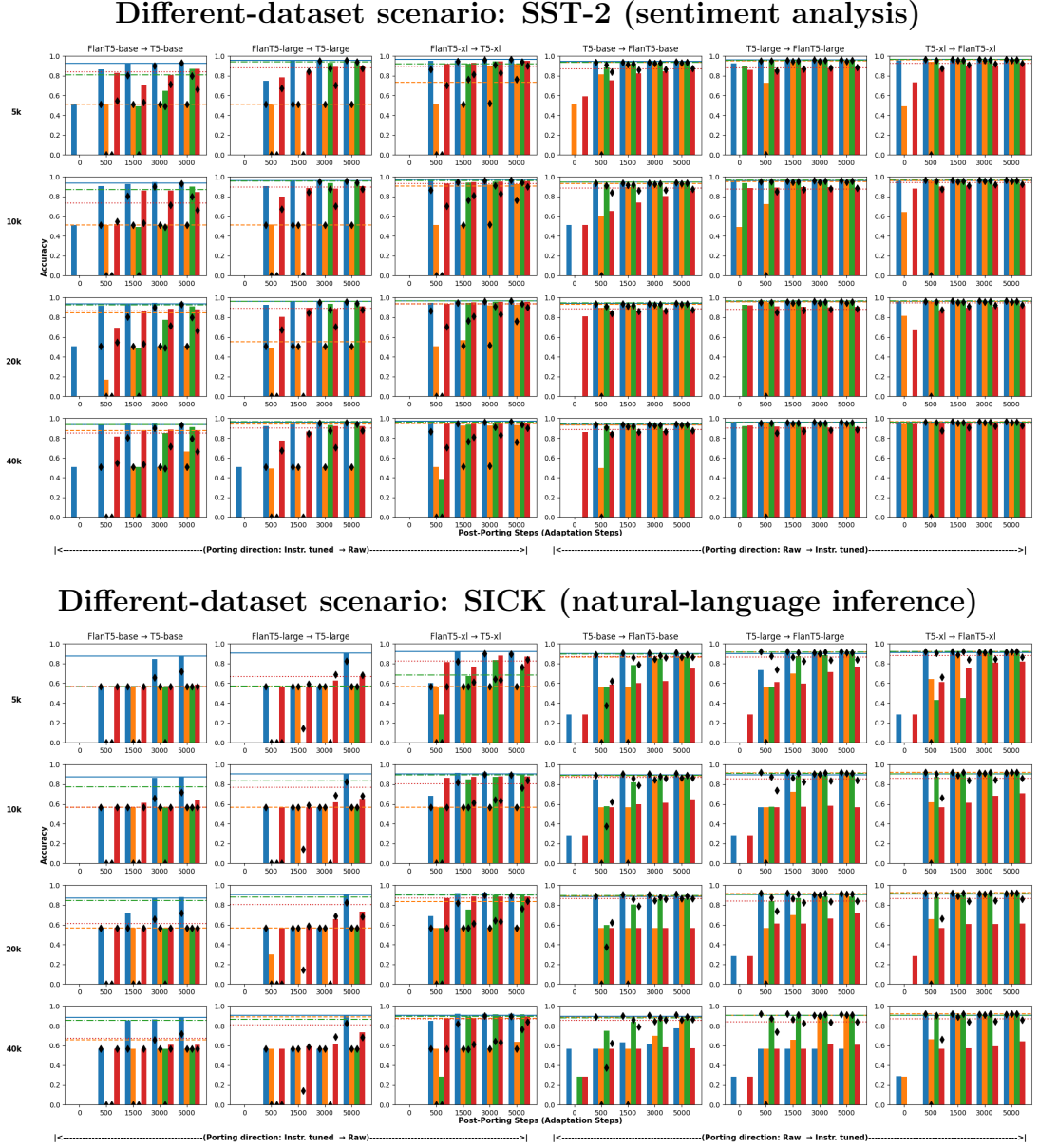


Figure 5.2: Performance of ported PEFT modules (continued): SST-2 and SICK.

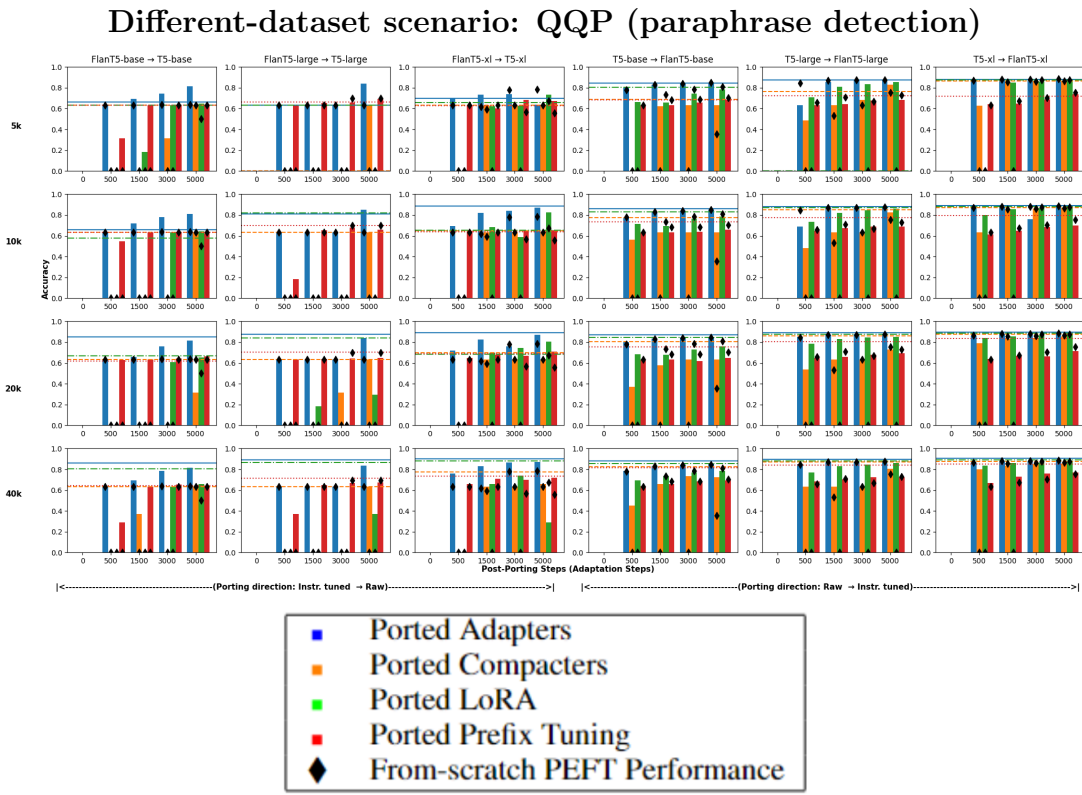


Figure 5.2: Performance of ported PEFT modules (continued): QQP and the shared legend.

High-level trends. Three patterns are visible in Figure 5.2. First, the strongest and most persistent gains appear in the sentiment-analysis and paraphrase-detection subfigures, especially Rotten Tomatoes, SST-2, MRPC, and QQP, where the coloured ported bars frequently sit above the black from-scratch diamonds across several post-porting budgets. By contrast, the SAMSum summarisation subfigure shows weaker and less consistent separation at small post-porting budgets, especially on the instruction-tuned→raw side, while the MNLI and SICK natural-language-inference subfigures are more mixed. Second, porting direction strongly governs zero-step behaviour: in the right-hand halves of several subfigures, especially Rotten Tomatoes, MRPC, SST-2, and QQP, the zero-step bars already show non-zero task performance, whereas in the left-hand halves zero-step non-collapse is rarer, and zero-step performance is often much lower. Third, same-task but different-dataset transfer is often competitive with same-dataset transfer: comparing Rotten Tomatoes with SST-2, MNLI with SICK, and MRPC with QQP, the different-dataset subfigures frequently show qualitatively similar portability gains, especially in the raw→instruction-tuned direction.

5.5.3 Zero-shot non-collapse rates

Zero-shot evaluation considers the importing host immediately after module import, with no post-porting updates. Because the reported criterion here is simply non-zero task performance, I treat this measure as *zero-shot non-collapse* rather than as a strong test of portability success. It is useful as a weak indicator that importing does not completely destroy task behaviour, but it should not on its own be interpreted as sufficient evidence of useful transferred functionality. Table 5.5 reports the frequency of zero-shot non-collapse across the full grid, using the criterion defined in Section 5.2.2.

The ‘Overall’ column reports the proportion of all evaluated cases in which the imported module achieves non-zero zero-step task performance. The two direction-specific columns split that same count by importing direction: instruction-tuned→raw

PEFT	Overall (%)	Instruction-tuned to Raw	Raw to Instruction-tuned
Houlsby-style Adapters	73 cases (43%)	22/84 (26%)	51/84 (60%)
Compacter	41 cases (24%)	12/84 (14%)	29/84 (34%)
LoRA	39 cases (23%)	12/84 (14%)	27/84 (32%)
Prefix Tuning	77 cases (45%)	12/84 (14%)	65/84 (77%)

Table 5.5: Rates of zero-shot non-collapse across models, tasks, and budgets.

and raw→instruction-tuned.

Two conclusions stand out. First, zero-shot non-collapse is common but technique-dependent: Houlsby-style Adapters (43%) and Prefix-Tuning (45%) achieve the highest overall rates, while LoRA (23%) and Compacter (24%) are lower. Second, non-collapse is direction-asymmetric: all techniques have higher zero-step non-collapse when porting raw→instruction-tuned than in the reverse direction, with Prefix-Tuning showing the strongest asymmetry (77% versus 14%). These findings are consistent with the qualitative direction effects already visible in Figure 5.2 and motivate the pretrained-knowledge analysis in Section 5.5.5.

5.5.4 Matched-budget portability criteria

Matched-budget comparisons test whether importing provides a performance advantage over training a new module from scratch under the same number of post-porting learning steps. Such an advantage is consistent with the possibility of adaptation-compute savings, but the present study does not directly measure steps-to-target, floating-point operations, or wall-clock savings. I therefore report the frequency with which a ported module (i) outperforms a from-scratch module under the same post-porting budget and (ii) recovers its exporting-side performance.

The definition of a configuration-level case is given in Section 5.2.2, while the repeated-seed aggregation used before counting cases is documented in Appendix C.2.4.

Criterion I: Outperforming from-scratch baselines under matched post-porting budgets

Table 5.6 reports the percentage of cases in which a ported module outperforms its matched from-scratch counterpart under the same post-porting learning-step budget. Figure 5.2 provides the corresponding fine-grained view: for each task setting, the black diamonds mark from-scratch performance and the coloured bars show ported-module performance under the same post-porting budget.

The ‘Overall’ column aggregates over all evaluated cases for a given PEFT technique. The budget columns then split the same comparison by post-porting learning-step budget, allowing the reader to see whether porting helps mainly at very small importing-side budgets, mainly after more adaptation, or relatively consistently across budgets.

Results and interpretation. Overall, Houlsby-style Adapters (58.92%) and Prefix-Tuning (54.76%) most frequently outperform from scratch. Compacter is intermediate (51.19%) but shows a particularly strong advantage at the smallest post-porting budget (0.5K: 75%) that diminishes with further importing-side adaptation, while LoRA is the least consistently beneficial (39.73%). The budget breakdown therefore suggests two different gain profiles: *early gains*, where importing is especially helpful at very small post-porting budgets, and *persistent gains*, where a ported module remains competitive or superior across a wider budget range.

Figure 5.2 shows where these two gain profiles arise. In several task settings, the gap between coloured bars and black diamonds is clearest at 0 or 0.5K post-porting steps and then narrows at 3K or 5K, which is the visual signature of early gains. In other settings, especially in several sentiment and paraphrase panels, the ported bars remain above the black diamonds across multiple post-porting budgets, which is the visual signature of more persistent gains.

PEFT	Overall	Post-porting learning steps			
		0.5K	1.5K	3K	5K
Houlsby-style Adapters	58.92%	54.76%	61.90%	57.14%	61.90%
Compacter	51.19%	75.00%	50.60%	35.12%	44.05%
LoRA	39.73%	35.71%	39.88%	42.86%	40.48%
Prefix-Tuning	54.76%	64.88%	51.79%	52.98%	49.40%

Table 5.6: Percentage of cases where ported modules outperform from-scratch modules under matched post-porting budgets.

Criterion II: Recovering exporting-side performance

Table 5.7 reports how often ported modules recover at least 99% of their exporting-side (pre-porting) performance after post-porting adaptation; that is, how often the ratio of importing-side post-porting performance to exporting-side pre-porting performance is at least 0.99 (Section 5.2.2).

The ‘Overall’ column reports the proportion of all evaluated cases in which a ported module recovers at least 99% of its exporting-side performance. The budget columns then split the same recovery criterion by post-porting learning-step budget, allowing the reader to see how quickly each PEFT technique regains its exporting-side level after import.

Results and interpretation. Recovery is strongly budget-dependent: for all techniques, the proportion increases monotonically with post-porting steps. Houlsby-style Adapters recover exporting-side performance most reliably overall (53.57%), rising from 24.40% at 0.5K steps to 74.40% at 5K. Prefix-Tuning is intermediate (38.10% overall), while LoRA is least likely to recover at small budgets (11.31% at 0.5K) but improves steadily with more adaptation (47.62% at 5K). Compacter also improves with budget, but remains below Houlsby-style Adapters throughout.

Figure 5.3 complements Table 5.7 by showing where recovery is fast or slow across tasks, porting directions, PEFT techniques, and budgets. Each heatmap cell reports the ratio between importing-side post-porting performance and the corresponding exporting-side pre-porting performance. Values near 100% indicate close recovery, while values above 100% indicate that importing-side performance exceeds

PEFT	Overall	Post-porting learning steps			
		0.5K	1.5K	3K	5K
Houlsby-style Adapters	53.57%	24.40%	50.00%	65.48%	74.40%
Compacter	35.57%	13.10%	33.33%	41.67%	54.17%
LoRA	28.72%	11.31%	20.24%	35.71%	47.62%
Prefix-Tuning	38.10%	23.21%	33.93%	44.64%	50.60%

Table 5.7: Percentage of cases where ported modules recover at least 99% of exporting-side performance after post-porting adaptation.

the exporting-side reference.

The heatmaps show that recovery is not distributed uniformly across the grid. Recovery tends to concentrate more strongly in some task families and porting directions than in others, and the raw→instruction-tuned direction more often approaches or exceeds full recovery at relatively small post-porting budgets. The figure therefore adds a task- and direction-sensitive view that is not visible in the aggregate percentages alone.

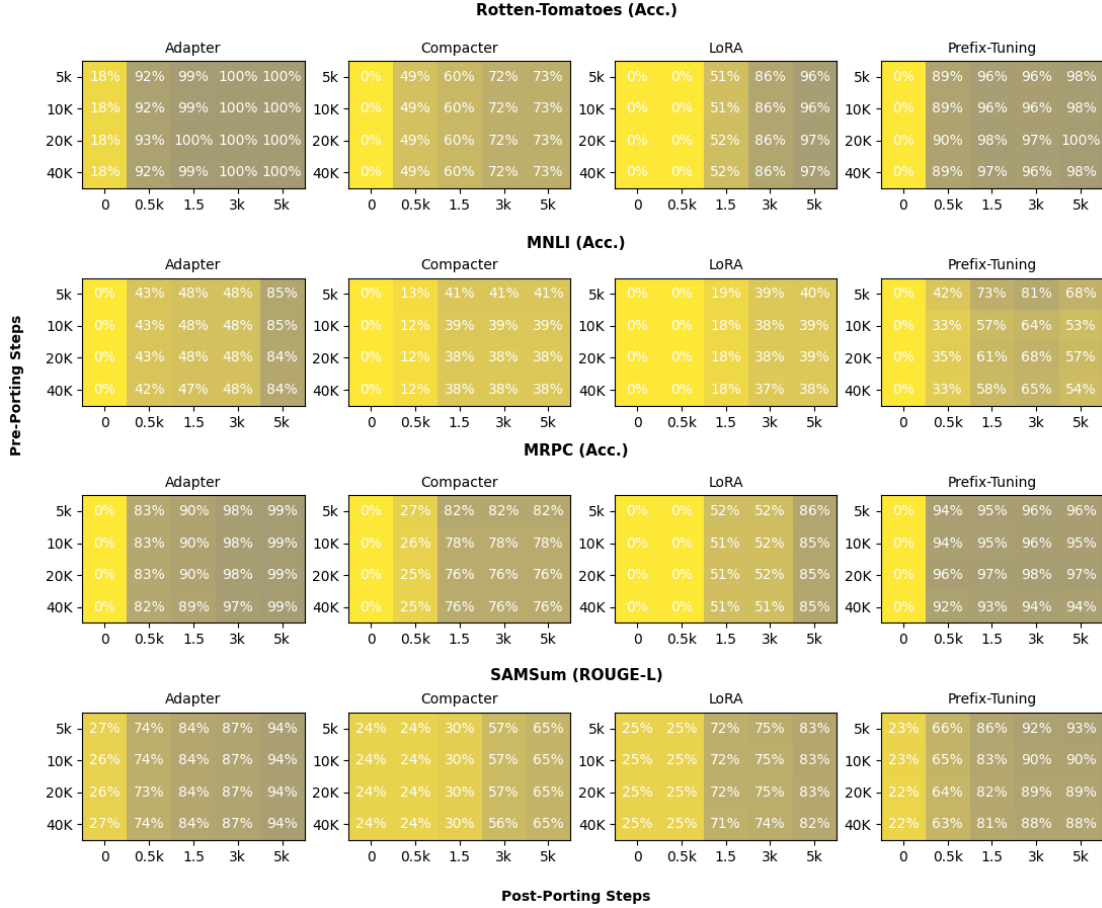


Figure 5.3: Recovery of exporting-side performance across pre- and post-porting budgets. Each cell is the importing-side score divided by the exporting-side score, averaged across models and two seeds. Values near 100% indicate near recovery and values above 100% exceed the exporting-side reference. The four panels are presented on separate pages at full width for readability. This panel shows same-dataset, instruction-tuned $\rightarrow$ raw transfer.

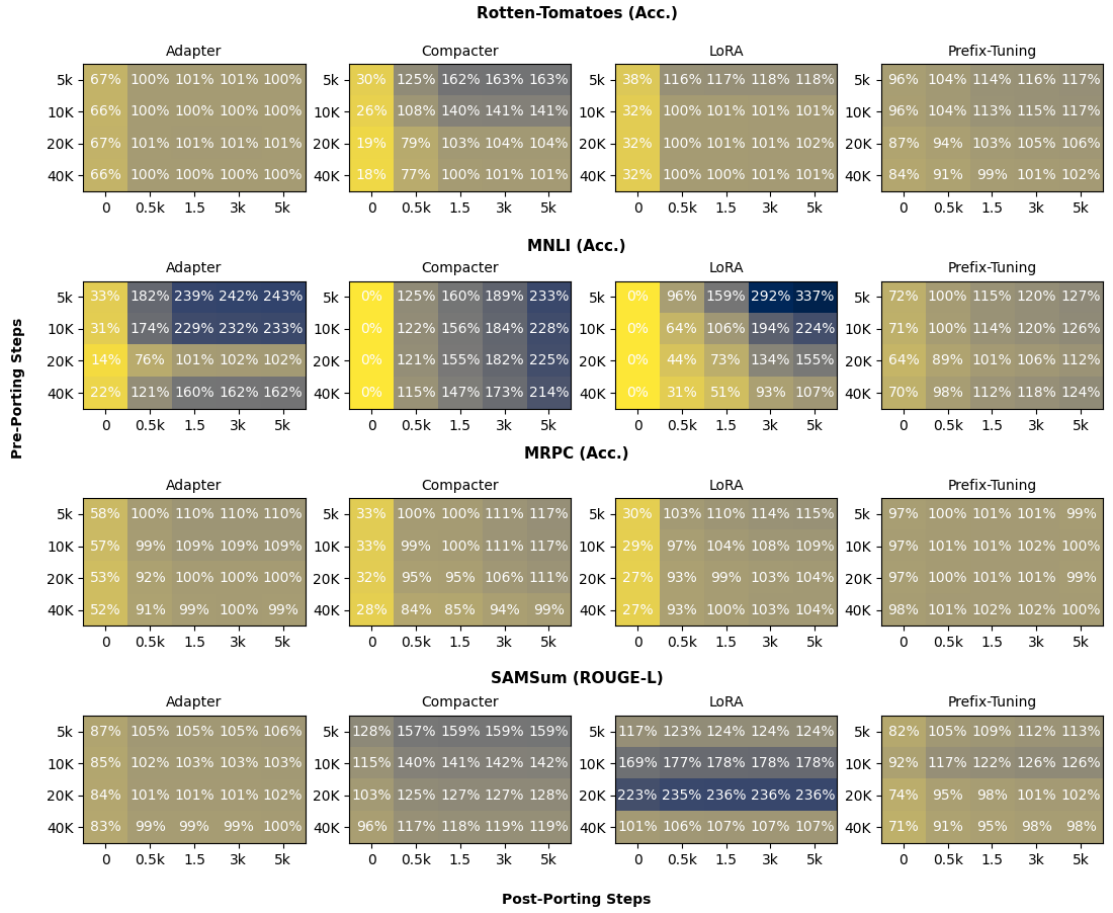


Figure 5.3: Recovery heatmaps (continued): same-dataset, raw $\rightarrow$ instruction-tuned transfer.

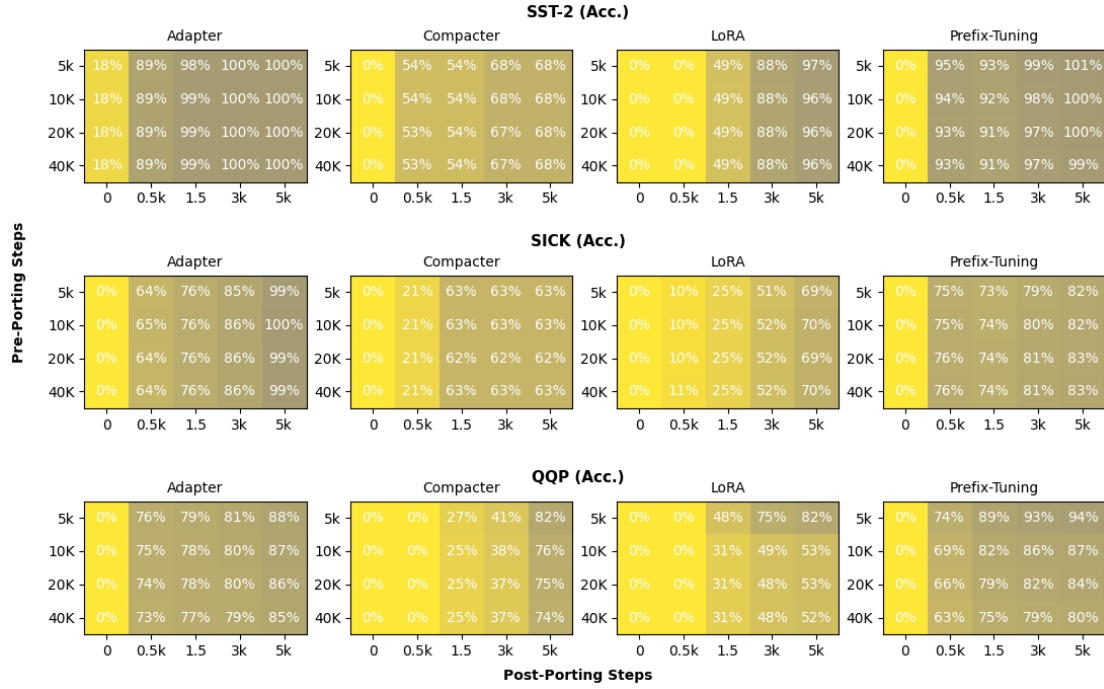


Figure 5.3: Recovery heatmaps (continued): different-dataset, instruction-tuned $\rightarrow$ raw transfer.

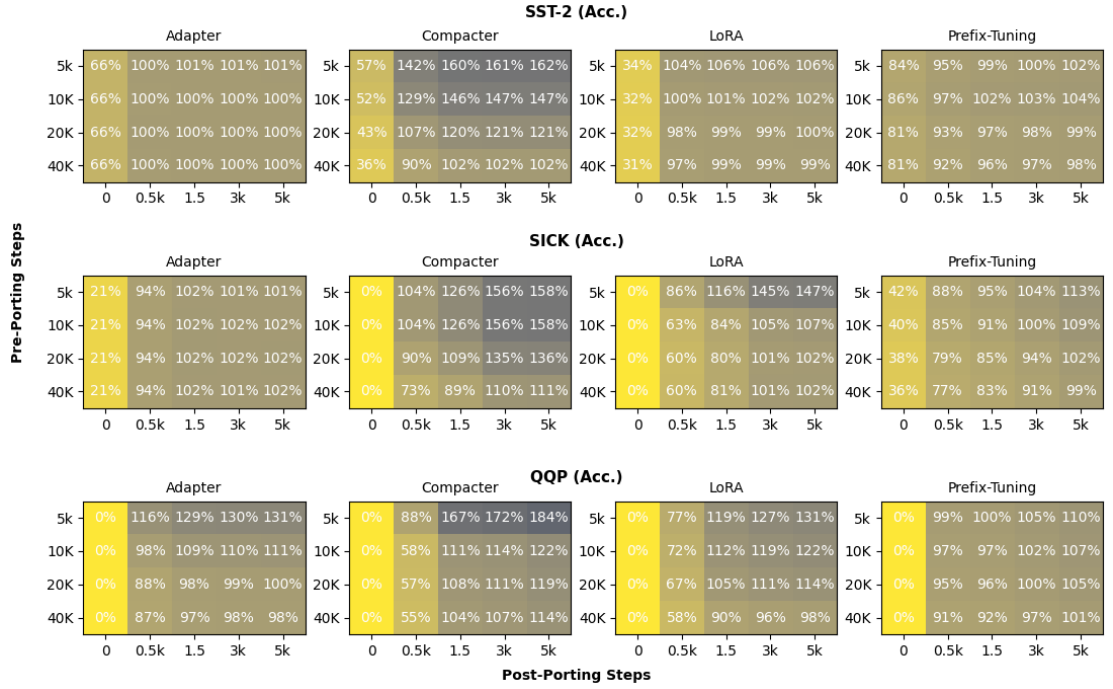


Figure 5.3: Recovery heatmaps (continued): different-dataset, raw $\rightarrow$ instruction-tuned transfer.

Criterion III: Satisfying both conditions

Table 5.8 reports the strictest condition used in the chapter: a ported module must both (i) outperform the matched from-scratch module under the same post-porting budget and (ii) recover at least 99% of its exporting-side performance.

The ‘Overall’ column reports the proportion of all evaluated cases in which both conditions are satisfied simultaneously. The budget columns split that joint criterion by post-porting learning-step budget, showing when a ported module both outperforms the matched from-scratch baseline and approaches its exporting-side level.

Results and interpretation. Under this joint criterion, successful cases generally become more frequent towards the higher-budget regime, although the trend is not monotonic for every technique. Houlsby-style Adapters are most reliable overall (36.76%), rising from 13.69% at 0.5K steps to 51.79% at 5K. Prefix-Tuning is second (30.21% overall) and also improves steadily with larger budgets. LoRA rises from 2.98% at 0.5K to 26.19% at 5K. Compacter shows a non-monotonic pattern, increasing from 12.50% at 0.5K to 18.45% at 1.5K, decreasing to 15.48% at 3K, and then reaching 25.60% at 5K.

This joint criterion is stricter than either of the two previous ones taken alone. A method can outperform from scratch without coming close to its exporting-side performance, and it can recover its exporting-side performance without clearly outperforming the matched from-scratch baseline. Requiring both conditions therefore isolates the strongest observed portability cases: cases where importing is beneficial relative to from-scratch training under the matched budget and also restores performance close to the module’s exporting-side level.

Under this joint criterion, qualifying cases are selective and become more frequent at larger post-porting budgets; within the evaluated grid, Houlsby-style Adapters and Prefix-Tuning satisfy the criterion most often.

PEFT	Overall	Post-porting learning steps			
		0.5K	1.5K	3K	5K
Houlsby-style Adapters	36.76%	13.69%	36.31%	45.24%	51.79%
Compacter	18.01%	12.50%	18.45%	15.48%	25.60%
LoRA	15.63%	2.98%	11.31%	22.02%	26.19%
Prefix-Tuning	30.21%	20.83%	28.57%	33.93%	37.50%

Table 5.8: Percentage of cases where ported modules both outperform from-scratch modules and recover at least 99% of exporting-side performance.

5.5.5 Pretrained task knowledge and porting direction

A key factor influencing portability is how much task-relevant information is already present in the host model prior to PEFT tuning. This is reflected in the recovery analysis relative to exporting-side performance (Table 5.7) and in the corresponding heatmap view (Figure 5.3).

Results and interpretation. Two patterns stand out. First, when the raw→instruction-tuned heatmaps are compared with the instruction-tuned→raw heatmaps, darker cells and near- or above-100% values appear more frequently on the raw→instruction-tuned side, especially for sentiment and paraphrase settings such as Rotten Tomatoes/SST-2 and MRPC/QQP. Second, this asymmetry is strongest at zero-shot and small post-porting budgets, but often remains visible even at 3K and 5K post-porting steps. These patterns suggest that PEFT tuning on a raw model may need to encode more of the task-specific behaviour inside the module itself, whereas instruction-tuned hosts already contain more relevant task behaviour, reducing how much additional task-specific behaviour must be carried by the imported module.

In the present study, the instruction-tuned hosts are Flan-T5 models trained on FLAN collections (Wei et al., 2022; Longpre et al., 2023). These collections include instruction-formulated training data closely aligned with several tasks in the evaluation suite, including SST-2, MNLI, QQP, MRPC, and SAMSum. Here, ‘overlap’ should be understood as overlap at the level of task type and, in several cases, direct task or dataset inclusion in the instruction-tuning mixture, rather than overlap of exact evaluation examples. This makes it plausible that instruction-

tuned hosts already contain part of the task-relevant behaviour needed for these evaluations, which in turn helps explain why modules ported into them often recover performance more readily than modules ported in the reverse direction.

5.6 Discussion: What governs portability?

The results presented above suggest that portability is governed by two interacting classes of factors: (i) **host- and task-level factors**, which determine what knowledge is already available and how demanding the task is, and (ii) **module-level factors**, which determine how the PEFT module interfaces with the host and what architectural compatibility it requires.

5.6.1 Host- and task-level factors

Porting direction strongly affects zero-step behaviour. Zero-shot non-collapse is considerably more frequent when transferring from raw→instruction-tuned hosts than in the opposite direction. A plausible explanation is that modules trained with a raw host must encode more task-specific behaviour inside the module itself, whereas modules trained with an instruction-tuned host can lean more heavily on task-relevant knowledge already present in the host model. When such a module is moved to a raw host, less of that behaviour is immediately available.

Task family and dataset setting shape how quickly ported modules help.

For some discriminative task settings, especially sentiment analysis and paraphrase detection, from-scratch modules on the importing host often catch up relatively quickly as post-porting budgets increase. For more complex settings, especially natural language inference, imported modules more often maintain an advantage across a wider budget range, which is consistent with a more persistent matched-budget advantage.

5.6.2 Module-level factors through the PEFT-Ref lens

Chapter 4 provides a descriptive scaffold for PEFT modules in terms of insertion form, workspace, integration form, and architectural repetition. In the present chapter, two module-level considerations are especially relevant: **architectural compatibility constraints** and **where the module intervenes**.

Compatibility and coupling to host-specific structure. All four modules repeat across Transformer layers (Table 5.3), so direct tensor import requires matching dimensions and a compatible layer mapping. These interface conditions determine whether import is mechanically feasible; they do not ensure that the same tensor has the same function in both hosts. The paired checkpoints used here also share a training lineage and thus a comparatively aligned parameter coordinate system. Independently trained same-architecture hosts may not provide that alignment.

The methods then differ in where they intervene: Houlsby-style Adapters and Compacters transform hidden representations sequentially, Prefix-Tuning injects context through attention keys and values, and LoRA reparameterises attention projections directly. In the evaluated grid, Houlsby-style Adapters and Prefix-Tuning have higher zero-shot non-collapse rates and more frequent matched-budget gains than LoRA; Compacter is more budget-dependent. One interpretation is that a direct weight reparameterisation is more sensitive to host-specific weights than context injection or representation-level transformations. The experiments do not isolate workspace from the other method differences, however, so this remains a hypothesis rather than a demonstrated mechanism.

Section summary. Taken together, the results suggest that portability depends jointly on what the host already knows, how demanding the task is, and how tightly the PEFT module is coupled to host-specific computation through its architectural interface.

5.7 Implications for budgeted adaptation in Chapter 6

Within corresponding T5 v1.1/Flan-T5 checkpoint pairs, the results show heterogeneous portability. Some imported modules retain a matched-budget advantage, but the frequency depends on technique, direction, task/dataset setting, and post-porting budget. The finding is therefore graded and setting-specific; it should not be generalised to independently trained hosts without further experiments.

From the thesis perspective, this is a functional test beyond Chapter 4: the present chapter asks whether modules retain utility after a controlled move to a paired checkpoint. A positive case indicates that some task-relevant information survives beyond the exporting checkpoint under the stated interface and lineage conditions.

This motivates the next question in Chapter 6: how limited adaptation computation can be allocated when deployment constraints are explicit. The transition does not require portability to hold universally; it uses the observed budget sensitivity as motivation for studying resource organisation directly.

The results here also motivate the use of *training-based transfer* as a complementary mechanism. When direct module reuse is infeasible because of architectural mismatch, or when budgets are too tight for imported modules to remain advantageous, distilled objectives can instead transfer behaviour into a constrained PEFT parameterisation on a chosen host model. Chapter 6 therefore uses a distillation-based testbed in which a smaller student is trained to mimic a stronger teacher while adaptation is restricted to budgeted LoRA modules, providing a controlled setting in which to study resource organisation under explicit constraints.

5.8 Summary

This chapter operationalised *portability* by asking whether PEFT modules retain useful task-specific behaviour when transferred between corresponding T5 v1.1/Flan-

T5 checkpoints under matched post-porting budgets. In the controlled sentiment study, ported modules have higher aggregate mean accuracy than the from-scratch and distribution-matched conditions; the latter control shows that tensor-wise means and variances alone do not reproduce the result. Across four task families and seven datasets, outcomes vary with PEFT technique, direction, task/dataset setting, and post-porting budget.

Taken together, the experiments measure a graded and budget-sensitive form of reuse within an aligned checkpoint family. Architectural and tensor-shape compatibility are necessary for direct import but are not sufficient in general; the study does not cover independently trained hosts. Chapter 6 next treats adaptation as a resource-organisation problem under budgets, while Part III uses intervention-based tests for the distinct internal-circuit setting.

Chapter 6

Budgeted Adaptation: Constraining LoRA Components via Budget-Constrained Optimisation

Chapters 4 and 5 developed the structural and paired-checkpoint portability analyses of PEFT modules in Part II. This chapter asks the next question: once adaptation must operate under explicit constraints, how can limited adaptation capacity be organised?

I address that question through a Low-Rank Adaptation (LoRA)-based study of *budgeted adaptation*. The study imposes a shared dense-compute schedule across LoRA-equipped projections and measures the resulting quality and cost-proxy operating points. I use teacher–student distillation as the controlled setting, transferring behaviour from a stronger teacher into a smaller student under an explicit dense-compute budget.

Within the broader thesis, this chapter therefore adds the resource-organisation step to the external PEFT-module story developed in Part II. At the larger teacher scales, the budgeted variants have higher mean ICL-probe scores than KD LoRA

within the reported warm-started setup. This setup-specific observation motivates the separate internal-structure analysis developed in Chapter 7.

Chapter outline. Section 6.1 motivates budgeted adaptation as the next step after portability and clarifies why distillation is the controlled transfer setting used in this chapter. Section 6.2 positions the chapter relative to prior work on knowledge distillation, LoRA-based student adaptation, adaptive rank allocation, and dense-path suppression. Section 6.3 presents the method in full detail, including the student-construction regime, the budget-aware gated LoRA modules, the budget controller, and the post-training compression procedure. Section 6.4 describes the models, data, training regime, evaluation measures, function-style ICL protocol, and compared methods. Section 6.5 presents the main results. Section 6.6 interprets those results and analyses the coordination between dense retention and low-rank usage. Section 6.7 discusses the broader meaning of the findings for PEFT modules, constrained adaptation, and the bridge to Part III. Section 6.8 draws out the implications for reusable PEFT modules and for the study of internal circuits. Section 6.9 summarises the chapter.

6.1 From portability to budgeted adaptation

Chapter 5 found matched-budget transfer advantages for some modules within paired T5-family checkpoints. The present chapter studies a different setting in which adaptation itself must operate under explicit deployment constraints: how limited capacity is organised between dense and low-rank computation.

This chapter studies that question through LoRA (Hu et al., 2021). As discussed in Chapter 4, LoRA is structurally attractive because it introduces a distinct low-rank adaptation pathway while leaving the dense host frozen. However, that same structural form also makes its limitation clear: standard LoRA reduces trainable parameter count without directly reducing dense inference computation. Uniform low-rank insertion therefore does not determine which dense projections should re-

main active, which can be removed or approximated, or where low-rank capacity should be concentrated across attention q/k/v/o and MLP up/gate/down projections.

The methodological question is therefore to treat LoRA as a *resource-organisation problem*: under an explicit dense-compute budget, how does a cost-based controller distribute computation across retained dense and low-rank pathways? The implemented controller does not measure behavioural importance, so the experiments evaluate a cost-only allocation rule rather than an importance-aware solution to this general problem.

I study this question in a controlled *distillation* setting. As discussed conceptually in Chapter 2, this thesis distinguishes direct module transfer from teacher-guided behavioural transfer. The present chapter uses the latter: the training objective combines next-token supervision from the corpus with teacher supervision that matches the teacher’s output distribution. Distillation therefore provides the controlled transfer setting for the budgeted-adaptation study.

The experiments compare sampled operating points under a single global budget parameter and describe how dense retention and low-rank usage change with that parameter. These points expose within-method quality–cost trade-offs, but they are not claimed to form a Pareto frontier or to select projections according to behavioural importance. The analysis is therefore bounded to the tested controller, student construction, teacher scales, and cost proxies.

In the fixed warm-started 6-layer-student setup, the budgeted variants score above KD LoRA on the function-style ICL composite at the larger teacher scales. Because all LoRA variants freeze the copied dense backbone and the final compression changes the parameterisation, this result may be specific to the initialisation and adaptation constraints. It motivates, but does not establish, a claim about internal structure; Chapter 7 examines causal reliance more directly.

6.2 Positioning relative to prior work

This chapter lies at the intersection of knowledge distillation, LoRA-based student adaptation, adaptive rank allocation, and dense-computation suppression. It studies one LoRA-based instantiation of budgeted adaptation in a distillation setting and compares it with full-parameter KD and fixed-rank KD LoRA.

Knowledge distillation for language models. As discussed conceptually in Chapter 2 and methodologically in Chapter 3, distillation is treated in this thesis as teacher-guided behavioural transfer from a stronger model to a smaller student. For language models, DistilBERT (Sanh et al., 2020a) showed that such transfer can produce substantially smaller deployment-oriented students while retaining much of the teacher’s performance. TinyBERT (Jiao et al., 2020) extended this with multi-stage distillation over embeddings, hidden states, and attention patterns, demonstrating that richer teacher supervision can improve compact-student quality. MiniLM (Wang et al., 2020a) further showed that relation-level attention transfer can yield strong compact models without reproducing the full teacher architecture. These methods establish distillation as a practical route to deployment-efficient students, but they generally assume a fixed student architecture and do not explicitly control how computation is allocated *within* the student once that architecture has been chosen.

LoRA and parameter-efficient distillation. LoRA (Hu et al., 2021) freezes pretrained dense weights and introduces trainable low-rank updates, making it an appealing parameterisation for parameter-efficient student adaptation. Recent work has combined LoRA with KD-style objectives in order to reduce the cost of student adaptation. For example, one line of work uses hybrid KD-LoRA training to keep teacher-supervised student adaptation parameter-efficient while retaining the basic LoRA structure (Azimi et al., 2024). A related line explores PEFT-based LLM distillation more broadly, again showing that teacher-guided transfer can be carried out

without full student fine-tuning (Yang et al., 2025). These methods are successful on their own terms in showing that LoRA-style distillation is viable and cheaper to train than full-parameter student tuning. However, they mainly address *training*-side efficiency: because the dense backbone remains intact, they do not directly solve the inference-side question of how dense computation itself should be reduced or reorganised.

Adaptive low-rank allocation. A separate line of work asks how low-rank capacity should be distributed across modules. AdaLoRA (Zhang et al., 2023), for example, argues that a single global rank is suboptimal and dynamically reallocates rank under a rank budget. This is important because it shows that low-rank capacity need not be distributed uniformly and that allocation can improve parameter efficiency relative to fixed-rank LoRA. However, methods of this type typically optimise rank or parameter allocation *alone*. They do not jointly control dense computation and low-rank capacity, and therefore do not directly address the deployment question that motivates this chapter.

Dense suppression, pruning, and progressive compression. A partially overlapping literature addresses the complementary problem of reducing dense computation directly through pruning, structured sparsification, or progressive suppression of dense pathways (Wen et al., 2016; Liu et al., 2017; Louizos et al., 2018; Sanh et al., 2020b; Muralidharan et al., 2024). These methods target dense compute more directly than LoRA-style adaptation, typically by learning which weights, channels, blocks, or structures can be removed with limited quality loss. PC-LoRA (Hwang et al., 2024) is especially relevant because it progressively suppresses the dense pathway during training to enable low-rank-only deployment. Its controller uses a single global decay schedule that drives dense computation toward zero uniformly. It therefore studies progressive dense-path suppression, whereas the implementation evaluated here studies how dense retention and low-rank paths change under one shared budget. This distinction describes the respective mechanisms; it

does not establish that the present cost-only controller improves on PC-LoRA or on importance-aware pruning methods.

Positioning of the present work. Relative to the cited methods, the implementation combines three features: a shared budget covering dense retention and low-rank paths, projection-level retention coefficients rather than a single uniform dense-decay coefficient, and a post-training conversion with dropped, SVD-approximated, or retained dense cases. AdaLoRA allocates low-rank capacity but does not jointly budget dense retention, while PC-LoRA progressively suppresses a dense path using a global schedule. The present comparison does not demonstrate superiority to the full pruning literature, and its cost-only removal rule should be understood as a simple baseline for future importance-aware variants.

6.3 Method: budgeted adaptation with LoRA

Figure 6.1 summarises the method at a high level. The purpose of this section is to make that budgeted-adaptation scheme precise by specifying the student-construction regime, the gated LoRA modules, the dense-budget controller, and the post-training compression procedure used in the reported experiments.

6.3.1 Student construction and distillation objective

As discussed methodologically in Chapter 3, I use distillation here as a controlled form of behavioural transfer instead of direct module portability. It provides a setting in which budgeted adaptation can be analysed after useful behaviour has been learned by the teacher. The student objective combines next-token supervision from the corpus with matching to the teacher’s output distribution.

Student construction. I use a fixed 6-layer student in all experiments. The student is constructed from the 12-layer 0.13B initialisation checkpoint by selecting a subset of teacher layers and copying them into a new 6-layer student before distil-

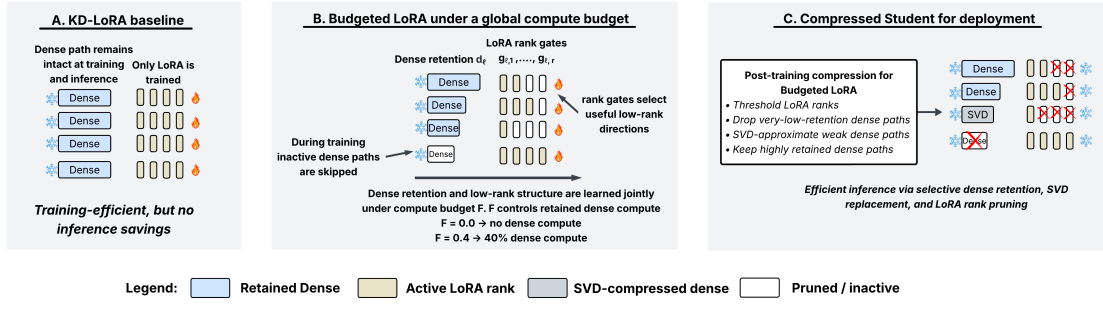


Figure 6.1: Budgeted adaptation with LoRA as resource organisation during distillation. Left: KD LoRA adds low-rank updates but leaves the dense backbone structurally unchanged at inference. Middle: the proposed method introduces a global dense-compute budget, realised through module-level dense-retention coefficients together with learned rank gates, so that computation is explicitly allocated between retained dense pathways and low-rank pathways during training. Right: after training, the student is consolidated into a deployment-efficient form by removing, approximating, or retaining dense components according to the resulting allocation profile.

lation training begins. The warm start also copies the token embeddings and final normalisation layer from the same checkpoint, and copies the LM head when the shapes match.

The ablation study in Appendix D.1.1 compares four student-construction modes: *truncated* (a contiguous prefix), *middle* (a contiguous middle block), *last* (a contiguous suffix), and *mixed* (a non-contiguous build spanning early, middle, and late depth). The mixed build performs best and is therefore used throughout. Concretely, the final student is built by selecting teacher layers $[0, 1, 3, 5, 7, 11]$ and copying them into the 6-layer student in that order. In the LoRA ablation, the same qualitative pattern holds as in full-parameter KD: mixed remains best (31.14 PPL), followed by middle (41.52), last (44.72), and truncated (48.09), so the student-construction choice is held fixed in the later budgeted-adaptation comparison (Appendix D.1.1; see also Appendix D.1.2 for the LoRA-side transfer of this choice).

Distillation objective. Following the standard teacher–student formulation of knowledge distillation (Hinton et al., 2015), let z_t^T and z_t^S denote teacher and student

logits at next-token prediction step t , and let

$$p_T^\tau(t) = \text{softmax}\left(\frac{z_t^T}{\tau_{\text{KD}}}\right), \quad p_S^\tau(t) = \text{softmax}\left(\frac{z_t^S}{\tau_{\text{KD}}}\right).$$

Let Ω denote the set of non-padding positions. The KD term is the average KL divergence from the teacher distribution to the student distribution over those positions:

$$\mathcal{L}_{\text{logit-KD}} = \frac{1}{|\Omega|} \sum_{t \in \Omega} \text{KL}(p_T^\tau(t) \parallel p_S^\tau(t)). \quad (6.1)$$

The full training objective is then a weighted combination of next-token cross-entropy and temperature-scaled teacher–student distribution matching:

$$\mathcal{L} = (1 - \lambda_{\text{KD}}) \mathcal{L}_{\text{CE}} + \lambda_{\text{KD}} \tau_{\text{KD}}^2 \mathcal{L}_{\text{logit-KD}}, \quad (6.2)$$

where $\mathcal{L}_{\text{CE}}$ is next-token cross-entropy with the corpus target. In the appendix ablation study, I first sweep temperature and KD mixing weight under a truncated warm-started student, then repeat the sweep for the strongest student-construction modes; the final reported setting is $\tau_{\text{KD}} = 3.0$ and $\lambda_{\text{KD}} = 0.8$, which is then held fixed across KD Full, KD LoRA, and the proposed method (Appendix D.1.1).

6.3.2 Budget-aware gated LoRA modules

Selected linear projections in the student are replaced by budget-aware gated LoRA modules. In the experiments reported here, LoRA is attached to all linear projections in each selected student block:

`q_proj, k_proj, v_proj, o_proj, gate_proj, up_proj, down_proj,`

yielding $7 \times 6 = 42$ adapted modules in total. For an input $x \in \mathbb{R}^{d_{\text{in}}}$ and base linear map $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, each gated module computes

$$y = d \cdot Wx + \frac{\alpha}{r_{\max}} B((Ax) \odot g), \quad (6.3)$$

where

- $A \in \mathbb{R}^{r_{\max} \times d_{\text{in}}}$ and $B \in \mathbb{R}^{d_{\text{out}} \times r_{\max}}$ are the LoRA factors;
- $r_{\max}$ is the maximum configured LoRA rank;
- $g \in (0, 1)^{r_{\max}}$ is a vector of learned per-rank gates;
- $d \in [0, 1]$ is the dense-path retention coefficient for this module;
- α is the LoRA scaling.

In the reported experiments, the final LoRA-based runs use $r_{\max} = 128$ and $\alpha = 256$. Rank 128 is the highest and best-performing value in the ablation over $r \in \{16, 32, 64, 128\}$ (Appendix D.1.2); it is used as a high-capacity fixed reference, not as evidence that 128 is universally optimal. In the Mistral-style setup used here, the adapted projections are bias-free. The dense weight W is frozen, while the LoRA factors and gate logits are trainable.

This parameterisation serves two purposes. First, the rank gates allow different modules to use different effective ranks rather than sharing one global low-rank bottleneck. Second, the dense retention coefficient controls how much of each dense projection remains active during training, making it possible to selectively and gradually reduce retained dense computation rather than committing to a hard structural change at initialisation. The result is a student whose trainable adaptation capacity and retained dense capacity can both vary across modules and over time.

6.3.3 Budget scheduling and controller

The dense budget is enforced by a controller operating over all gated modules. For each module m , I define a dense cost proxy

$$c_m = d_m^{\text{in}} d_m^{\text{out}}, \quad (6.4)$$

that is, the multiply-accumulate count of the dense linear projection, proportional to FLOPs up to a constant factor. The total retained dense cost at training step t is then

$$C(t) = \sum_{m=1}^M d_m(t) c_m, \quad (6.5)$$

where $d_m(t)$ is the current retention coefficient value for module m .

A target dense-budget fraction $b(t)$ is specified by a cosine schedule. Let t_0 and t_1 denote the start and end of the budgeting phase, expressed as fractions of total training steps, and let $F \in [0, 1]$ be the final dense fraction. Then

$$b(t) = \begin{cases} 1, & t \leq t_0, \\ F + (1 - F) \frac{1 + \cos\left(\pi \frac{t-t_0}{t_1-t_0}\right)}{2}, & t_0 < t < t_1, \\ F, & t \geq t_1. \end{cases} \quad (6.6)$$

The target dense cost is

$$C^\star(t) = b(t) \sum_{m=1}^M c_m. \quad (6.7)$$

The schedule ablation compares immediate removal, short decay, and long decay at $F = 0.0$. Scheduler behaviour depends on the number and timing of training tokens, and I did not establish a transfer law from short to full-budget runs; unlike most other ablations, this choice was therefore evaluated at the full 2.55B-token budget (Appendix D.1.3). Among the three tested schedules, the short decay gives the lowest validation perplexity while retaining more compute savings than the long decay:

$$(t_0, t_1) = (0.1, 0.3),$$

that is, the budget remains fully dense for the first 10% of training, reaches the final dense fraction F by 30% of training, and remains at F thereafter. This schedule

performs substantially better than both immediate collapse and a very long decay.

At each step, the controller enforces the scheduled dense-cost target by greedily lowering module-level retention coefficients until the retained dense cost satisfies the budget. In the reported experiments, modules are ordered by $s_m = c_m$ and reduced in ascending score order, so lower-cost modules are reduced first; ties follow deterministic module registration order. Updates are smoothed with an exponential moving average with coefficient $\beta = 0.2$, and values below the dense-removal threshold are clamped to zero.

This score contains *cost only*. It uses no validation-loss change, gradient, activation, weight magnitude, Fisher, or other measure of a module’s contribution to output quality. The controller is therefore importance-agnostic: it enforces a budget deterministically but cannot identify which projections are behaviourally least important. In the context of pruning and allocation work, it should be treated as a cost-based baseline. An importance-aware controller and a matched comparison between cost-only and importance-based removal are necessary future extensions; the present results cannot show that the selected modules are the best modules to remove.

In this chapter, *retention* refers to the training-time dense coefficient values $d_m(t)$, *allocation* refers to the resulting distribution of computation across retained dense pathways and low-rank pathways under the budget, and *removal* refers to the hard zeroing of dense paths once retention falls below threshold.

Pseudocode for the controller. Algorithm 1 summarises the training-time allocation procedure.

6.3.4 Post-training compression for efficient inference

During training, a dense path is still computed unless the controller has already clamped its retention coefficient to zero, while the low-rank path is still evaluated at the full configured rank. The raw training graph therefore does not yet realise

Algorithm 1 Budgeted LoRA training and controller update

Require: teacher T , student S , selected projections $\mathcal{M}$, final dense fraction F , schedule endpoints t_0, t_1 , KD parameters $\tau_{\text{KD}}, \lambda_{\text{KD}}$

```
1: for training step  $t = 1, \dots, T_{\text{train}}$  do
2:   Compute teacher logits  $z_T$  (no gradient) and student logits  $z_S$ 
3:   Optimise the student with Eq. (6.2)
4:   Compute the target dense fraction  $b(t)$  using Eq. (6.6)
5:   Compute the target dense cost  $C^*(t)$  using Eq. (6.7)
6:   Compute current retained dense cost  $C(t)$  using Eq. (6.5)
7:   if  $C(t) > C^*(t)$  then
8:     Order modules  $m \in \mathcal{M}$  by removal score  $s_m = c_m$  (ascending)
9:     for modules in that order do
10:      Lower  $d_m(t)$  by the amount needed to move  $C(t)$  toward  $C^*(t)$ 
11:      Smooth the update with EMA using  $\beta = 0.2$ 
12:      if  $d_m(t) < \epsilon_{\text{zero}}$  then
13:        Set  $d_m(t) \leftarrow 0$ 
14:      if  $C(t) \leq C^*(t)$  then
15:        break
16: Apply post-training compression (Algorithm 2)
```

the full inference savings implied by the learned gates. I therefore apply an explicit post-training compression pass before deployment.

Here, *compression* denotes the full post-training consolidation step, while *removal* refers specifically to the case where a dense path is dropped entirely. In the reported experiments, the compression ablation chooses the following thresholds:

- dense-removal threshold during training and compression:

$$\epsilon_{\text{zero}} = 10^{-3};$$

- LoRA gate hard threshold at compression:

$$\epsilon_{\text{gate}} = 0.3;$$

- dense-SVD threshold:

$$\epsilon_{\text{lr}} = 0.7;$$

- maximum SVD rank for residual dense approximation:

$$r_{\text{dense}}^{\text{max}} = 128.$$

The 10^{-3} dense-removal value is a numerical near-zero tolerance used to avoid retaining negligible scaled dense paths. The gate threshold 0.3 and dense-SVD threshold 0.7 are the best validation choices in the reported sweeps over $\{0.3, 0.5, 0.9\}$ and $\{0.5, 0.7, 0.9\}$, respectively (Appendix D.1.3). They are operational compression settings, not universal sparsity thresholds and not importance scores. The SVD rank cap of 128 matches the configured maximum LoRA rank so that the intermediate dense approximation does not exceed the low-rank capacity used elsewhere in the comparison.

For each gated linear module, I first harden the rank gates by thresholding at ϵ_{gate} and keep at least one active rank. I then choose one of three deployment cases based on the final retention coefficient value d .

Case 1: drop the dense path entirely. If $d < \epsilon_{\text{zero}}$, the dense path is removed and only the pruned low-rank component is kept. This yields a two-layer low-rank factorisation implementing the LoRA path alone.

Case 2: approximate the residual dense contribution by low-rank SVD. If $\epsilon_{\text{zero}} \leq d < \epsilon_{\text{lr}}$, the remaining dense contribution dW is treated as small enough to approximate with a low-rank SVD:

$$dW \approx U_k V_k. \quad (6.8)$$

The approximation rank k is chosen adaptively up to a configured maximum:

$$k = \max\left(1, \left\lfloor r_{\text{dense}}^{\text{max}} \cdot \frac{d}{\epsilon_{\text{lr}}} \right\rfloor\right), \quad (6.9)$$

clipped to at most $\min(d_{\text{out}}, d_{\text{in}})$. Thus, dense components with larger retained

Algorithm 2 Post-training compression for deployment

```
1: for each gated module  $m$  do
2:   Threshold rank gates at  $\epsilon_{\text{gate}}$ ; keep at least one active rank
3:   if  $d_m < \epsilon_{\text{zero}}$  then
4:     Drop the dense path; keep only the pruned low-rank factorisation
5:   else if  $\epsilon_{\text{zero}} \leq d_m < \epsilon_{\text{lr}}$  then
6:     Approximate  $d_m W_m$  by low-rank SVD of rank  $k$  from Eq. (6.9)
7:     Concatenate the SVD factors with the hardened LoRA factors
8:   else
9:     Merge retained dense path and LoRA update into  $W_{\text{eff}} = d_m W_m + \Delta W_{\text{LoRA}}$ 
```

coefficients receive higher SVD rank, while components closer to the removal threshold receive lower rank. The resulting dense approximation is then concatenated with the hardened LoRA factors, yielding a single fused low-rank module.

Case 3: keep the dense path and merge. If $d \geq \epsilon_{\text{lr}}$, the dense path is retained and merged with the hardened LoRA update into a standard dense linear layer:

$$W_{\text{eff}} = dW + \Delta W_{\text{LoRA}}. \quad (6.10)$$

This case is used when the module still requires substantial dense capacity for accuracy.

Pseudocode for compression. Algorithm 2 summarises the deployment conversion.

These three cases produce a deployment model that is materially different from the training graph. Compression can realise both rank pruning and dense removal, thereby translating training-time budgeting signals into actual inference-time savings.

6.4 Experimental setup

This section instantiates the budgeted-adaptation method from Section 6.3. It specifies the teacher–student model family, training corpus, student-side optimisation

budget, evaluation measures, and compared methods used in the experiments. Appendix D.1 reports the ablations used to select the final student construction, KD hyperparameters, LoRA configuration, and budget schedule; Appendix D.2 gives the full training-compute proxy derivation; and Appendix D.3 gives the complete ICL probe protocol and per-task results.

6.4.1 Models, student regime, and training data

Teacher and student models. All teacher and student models use a Mistral-style decoder-only Transformer with pre-norm blocks, rotary position embeddings, SwiGLU feed-forward layers, grouped-query attention, and head dimension 64 (Jiang et al., 2023). The checkpoints used in this chapter correspond to the *baseline* natural-only models that later form the starting point of the broader Bi-Induct family introduced in Chapter 7. In other words, Chapter 7 extends these baseline models with targeted synthetic data conditions in order to study load-bearing structure in in-context learning circuits; in the present chapter, however, I use only the baseline natural-text checkpoints.

I evaluate three teacher scales: 0.13B, 0.5B, and 1B. Their architectural specifications are summarised in Table 6.1. All three teachers were pretrained on THE PILE under the same compute-optimal regime and yield validation perplexities of 21.8, 16.0, and 14.1 respectively. The deployment student is always a fixed 6-layer model initialised from the 0.13B checkpoint; for the 0.5B and 1B settings, only the teacher scale changes. This creates a controlled ladder of teacher quality while holding student architecture fixed.

Model-creation regime and training budget. All teacher pretraining and student distillation use the same deduplicated THE PILE corpus (Gao et al., 2020), with shared tokenisation and sequence length 1024. I define a fixed held-out evaluation slice using a stable MD5-based hash, corresponding to 0.2% of the corpus (roughly 0.4B tokens), and all evaluation subsets are drawn from this slice.

Model	Layers	Hidden size	MLP width	Attn. heads / KV heads	Val. PPL ↓
0.13B teacher	12	768	3072	12 / 3	21.8
0.5B teacher	30	1024	4096	16 / 4	16.0
1B teacher	30	1536	6144	24 / 6	14.1

Table 6.1: Teacher-model presets used in this chapter. All teachers follow the same compute-optimal pretraining setup, while the deployment student is a fixed 6-layer model distilled from the selected teacher. *Hidden size* denotes the Transformer residual width, *MLP width* the feed-forward expansion dimension, *Attention heads / KV heads* the number of query heads and grouped key-value heads, and *Val. PPL* the validation perplexity on the held-out THE PILE slice.

As described in Chapter 3, pretraining throughout the thesis follows a Chinchilla-style compute-optimal regime, approximately 20 training tokens per parameter (Hoffmann et al., 2022). The checkpoints used here are the baseline natural-only models from that broader pretraining programme; fuller pretraining details for that model family are given later in Chapter 7.

For the chapter-local student comparison, all 6-layer student variants are trained for 2.55B tokens. This matches the original token budget of the 0.13B checkpoint from which the student is initialised. The purpose of this choice is not to equalise *total historical compute* between larger teachers and the student. Rather, it fixes the *student-side optimisation budget* across KD Full, KD LoRA, and Budgeted LoRA, so that differences in final behaviour are attributable to the adaptation method and budget organisation rather than to different amounts of student training.

All final distillation runs use two NVIDIA A100 GPUs with bf16 precision and per-device batch size 2^{14} tokens. Optimisation uses AdamW with base learning rate 3×10^{-4} , zero weight decay, and a cosine learning-rate schedule with 3% warmup capped at 2000 steps; gradients are clipped to a maximum norm of 1.0 (Loshchilov and Hutter, 2019). These settings are shared across KD Full, KD LoRA, and Budgeted LoRA unless otherwise stated.

6.4.2 Evaluation measures and reported quantities

Evaluation is deliberately multi-axis because a compressed student can preserve some properties while losing others. I therefore report four quantities before interpreting the results.

1. **Language-model quality.** I report held-out perplexity (PPL) on the same 10M-token validation set drawn from the held-out THE PILE slice. Lower PPL indicates better next-token modelling quality.
2. **Student-side training compute.** I report a compact training-compute proxy that counts the student-side dense and low-rank adapted-module computation used during distillation. This is a method-comparison proxy rather than measured wall-clock time or exact hardware FLOPs; the full derivation is given in Appendix D.2.
3. **Compressed-module inference efficiency.** I report inference speedup and parameter reduction for the replaced or compressed modules after post-training compression. These quantities describe the adapted-module portion of the student, not end-to-end serving speed for the whole model.
4. **In-context learning retention.** I report a macro-averaged ICL composite over the 19-task function-style probe suite adapted from Todd et al. (2024). The main chapter reports the aggregate composite, while Appendix D.3.1 gives the complete task and prompting protocol and Appendix D.3.2 gives the task-level breakdown.

6.4.3 Function-style ICL probes

The ICL evaluation uses the 19 function-style probes adapted from the Function Vectors evaluation suite. These tasks are deliberately simple but context-dependent: they require the model to infer an input-output mapping from demonstrations rather than rely on stored facts. The suite spans six categories:

-
- **Alphabetic ordering:** selecting the alphabetically first or last item from sets of size 3 or 5;
 - **Selection probes:** choosing the first, middle, or last item from sequences of length 3 or 5;
 - **Capitalisation transformations:** capitalising full words, first letters, last letters, or predicting the next capital letter;
 - **Lowercasing transformations:** lowercasing first or last letters;
 - **Sequence continuation:** predicting the next or previous item;
 - **Simple symbolic properties:** predicting a basic property such as word length.

These probes are appropriate here because they depend strongly on using the provided context, thereby reducing confounds from alternative pathways such as memorised facts or parametric recall.

ICL prompting and output extraction. I use a fixed 10-shot prompting protocol across all models and average over three random demonstration seeds. Each in-context example is formatted as **Q:**<input> followed by **A:**<output>, with a new-line between question and answer and a blank line between demonstrations. No additional instruction prefix is used. The query is appended in the same format and terminates at **A:**, requiring the model to generate the target continuation. Outputs are evaluated using HITS@1 exact-match accuracy after canonicalising the generated continuation to the target string for the corresponding probe family. In the main text I report the macro-averaged ICL composite; the full task-level breakdown remains in Appendix D.3.2.

6.4.4 Compared methods

I compare two knowledge-distillation (KD) baselines against the proposed method, while keeping student construction, corpus, and student-side token budget fixed.

KD Full. This baseline trains the 6-layer student with full-parameter distillation and serves as the strongest quality-oriented reference. The mixed student build is used throughout. Training uses the weighted combination of next-token cross-entropy and logit-based KD from Eq. (6.2). The student-construction choice and the final KD hyperparameters are selected through the ablations in Appendix D.1.1. Following those ablations, I use $\tau_{\text{KD}} = 3.0$ and $\lambda_{\text{KD}} = 0.8$.

KD LoRA. This baseline keeps the same 6-layer warm-started student construction, but replaces full student tuning with LoRA-based parameter-efficient distillation. LoRA is applied to all linear projections in each selected Transformer block, yielding 42 adapted modules in total. Based on the LoRA ablation study in Appendix D.1.2, I fix the rank to $r = 128$, the scaling to $\alpha = 256$, and the dropout to 0.0. As in KD Full, I use $\tau_{\text{KD}} = 3.0$ and $\lambda_{\text{KD}} = 0.8$, chosen via Appendix D.1.1.

Budgeted LoRA. Budgeted LoRA builds directly on KD LoRA and adds budget-aware gated modules, dense-retention coefficients, and learned rank gates under a shared schedule. The core KD and LoRA hyperparameters are held at $\tau_{\text{KD}} = 3.0$, $\lambda_{\text{KD}} = 0.8$, $r = 128$, $\alpha = 256$, and dropout 0.0; the schedule and compression settings are chosen in Appendix D.1.3.

The reported budgets are selected as interpretable sampled cases. $F = 0.0$ is the boundary case in which no dense cost is retained at the final schedule value, while $F = 0.4$ is a moderate case that yields perplexity close to KD LoRA in the reported runs. $F = 0.8$ is used later only as a higher-retention structural reference. These values do not exhaust the budget interval, establish optimal settings, or imply that the reported points are Pareto-optimal.

6.5 Results

This section reports held-out perplexity and student-side cost proxies, followed by the function-style ICL composite. The comparisons are sampled operating points

Method	Fixed 6-layer student PPL ↓			Efficiency (student-side)		
	0.13B teacher	0.5B teacher	1B teacher	Train compute ↓	Infer. speedup ↑	Param. red. ↑
<i>Baselines</i>						
KD Full	20.88	18.26	17.86	1.00×	1.00×	0%
KD LoRA	24.32	26.06	26.50	0.91×	1.00×	0%
<i>Budgeted LoRA</i>						
$F = 0.4$	24.34	26.19	26.58	0.59×	1.74×	53.9%
$F = 0.0$	25.13	28.45	28.86	0.38×	4.05×	80.2%

Table 6.2: Quality–cost comparisons across distillation methods. Every quality entry evaluates the same fixed 6-layer deployment-student architecture; the three columns vary *teacher scale*, not deployed model size. Training compute is an adapted-module proxy normalised to KD Full and excludes teacher cost. Inference speedup and parameter reduction cover the replaced/compressed modules only, not the full student.

within the fixed student setup.

6.5.1 Quality–efficiency trade-offs

I first compare validation perplexity with the adapted-module training-compute proxy and post-compression module-level speedup and parameter reduction. The efficiency values are not end-to-end wall-clock or serving measurements.

KD Full is the strongest perplexity baseline in every teacher column. At the same teacher scale, Budgeted LoRA with $F = 0.4$ is close to KD LoRA in perplexity (differences of 0.02, 0.13, and 0.08) while its proxies report lower adapted-module training cost and a smaller compressed module footprint. The $F = 0.0$ case reports the largest proxy savings and the highest perplexity. These are within-setting trade-offs, not a claim of Pareto dominance.

The smaller-teacher fully trained baseline is an important cross-column comparison. KD Full with the 0.13B teacher obtains 20.88 PPL, whereas Budgeted LoRA distilled from the larger 0.5B and 1B teachers obtains 26.19/26.58 at $F = 0.4$ and 28.45/28.86 at $F = 0.0$. Thus, the larger-teacher budgeted variants do not beat the 0.13B-teacher KD Full student in absolute perplexity. Since all columns deploy the same 6-layer architecture, this is a hard dense-student baseline rather than evidence

that pruning a larger deployed student is advantageous. The supported advantage is narrower: relative to KD LoRA with the *same teacher*, $F = 0.4$ retains similar perplexity while reducing the reported adapted-module cost proxies.

Training-compute proxy used in Table 6.2. Because the quality–efficiency table is central to the chapter, I define the training-cost proxy in the main text rather than deferring it entirely to the appendix. As in Appendix D.2, this proxy is defined at the level of the adapted/replaced modules rather than full end-to-end system FLOPs. Let D denote the forward cost of a dense adapted projection and L the forward cost of its low-rank path. I use a first-order approximation in which forward and backward passes have comparable cost. Then:

$$C_{\text{full}} \approx 3D, \tag{6.11}$$

$$C_{\text{LoRA}} \approx 2D + 3L, \tag{6.12}$$

$$C_{\text{bud}} \approx 2\bar{d}D + 3L, \tag{6.13}$$

where $\bar{d}$ is the average retained dense fraction over training. Under the short-decay schedule used here, $\bar{d} \approx 0.2 + 0.8F$. With the instantiated values from the adapted modules, this yields $C_{\text{LoRA}}/C_{\text{full}} \approx 0.91$, $C_{\text{bud},0.0}/C_{\text{full}} \approx 0.38$, and $C_{\text{bud},0.4}/C_{\text{full}} \approx 0.59$. This proxy measures scheduled retained dense contribution rather than exact hardware FLOPs, so it should be read as a student-side allocation model for comparing methods rather than as measured wall-clock time. The full derivation and instantiated values are given in Appendix D.2.

The table therefore reports two sampled budget settings with different quality and proxy-cost outcomes. It does not establish an optimal frontier, and KD Full remains the stronger quality baseline.

6.5.2 Retention of in-context learning behaviour

Perplexity alone does not measure the behaviours covered by the ICL probe suite. In the broader thesis framing established in Chapter 2, in-context learning is a form

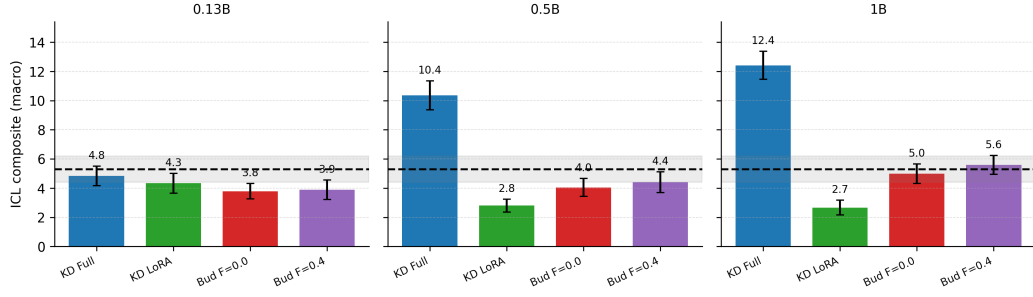


Figure 6.2: Macro-averaged ICL composite score over the 19 function-style tasks for students distilled from teachers at three scales (0.13B, 0.5B, and 1B). Bars show the mean across tasks and three random demonstration seeds, with error bars indicating variability across demonstration seeds. The dashed horizontal line marks the ICL composite score of the full 0.13B model used as the student initialisation source.

of reuse without parameter updates: the model must use context as computation in addition to predicting the next token accurately. I therefore evaluate macro-averaged in-context learning (ICL) performance on the 19-task function-style probe suite; the task families and prompting protocol are summarised in Section 6.4.3 and documented in full in Appendix D.3.1. Chapter 7 later examines the corresponding internal load-bearing structure directly.

The figure reports the macro-averaged ICL composite across the 19 function-style probes for each method and teacher scale. The dashed line marks the full 0.13B model from which the student is initialised. The key question is not absolute competence on these probes, which remains low for these small models, but relative retention of context-dependent behaviour across methods and teacher scales. Appendix D.3.1 gives the full task and prompting details, and Appendix D.3.2 reports the complete per-task breakdown.

Observed mean differences. KD Full’s macro composite increases from 4.8 with the 0.13B teacher to 12.4 with the 1B teacher, whereas KD LoRA records 2.8 and 2.7 with the 0.5B and 1B teachers. The corresponding Budgeted LoRA means are higher: at 1B, 5.0 for $F = 0.0$ and 5.6 for $F = 0.4$. These values average 19 heterogeneous tasks and three demonstration seeds; they are descriptive probe results, not a statistical demonstration of general ICL improvement. The per-task breakdown in Appendix D.3.2 shows substantial task variation.

What these probes test. Success on these tasks depends on inferring a simple mapping from the demonstrations in context. For example, the model may need to choose the middle item from a list, apply a letter-case transformation, or continue a short symbolic sequence. The demonstrations specify the required mapping. Probe performance therefore tests whether the student uses the prompt as computation beyond modelling local token statistics.

Within this specific warm-started, frozen-backbone setup, the budgeted variants have higher ICL-composite means than KD LoRA at the larger teacher scales. This comparison is not evidence that budgeted compression generally preserves ICL better across model families or student initialisations.

6.6 Interpretation and analysis

This section interprets the sampled quality–cost comparisons, the relation between perplexity and the ICL probe composite, and the final compressed structures. Because the controller is cost-only and the experiment does not include an importance-aware comparator, the interpretation is descriptive rather than causal.

6.6.1 Interpreting the observed operating points

Table 6.2 samples two values of the global budget. At $F = 0.4$, the perplexity is close to KD LoRA with the same teacher and the adapted-module proxies are lower; at $F = 0.0$, the proxies decrease further and perplexity worsens. This demonstrates that the implemented control changes the quality–cost balance within the method. It does not show that intermediate values form a continuous curve, that the points are Pareto-optimal, or that they dominate KD Full.

6.6.2 Perplexity, teacher scale, and ICL retention

One useful interpretation of the perplexity results is a distillation capacity gap. As teacher scale increases, the supervision signal becomes richer. KD Full can exploit

stronger teachers because the entire student can reorganise. KD LoRA, by contrast, can adapt only through a restricted low-rank subspace around a frozen backbone, and therefore does not benefit from larger teachers in the same way. Budgeted LoRA still follows the broad perplexity trend of KD LoRA, which suggests that its gains do not arise from removing the low-rank bottleneck at the level of distribution matching alone.

The ICL composite is not ordered in the same way as perplexity: at larger teacher scales, Budgeted LoRA and KD LoRA have similar perplexity but different mean probe scores. One possible explanation is that the training and compression parameterisation affects behaviours that the aggregate language-modelling metric does not capture. However, the setup also offers a narrower explanation. KD LoRA and Budgeted LoRA both start from the same selected 6-layer copy, freeze the dense weights, and use rank 128; Budgeted LoRA additionally scales and sometimes replaces dense paths before post-training consolidation. The copied-and-frozen construction may constrain KD LoRA, while the altered effective parameterisation may change optimisation freedom. No alternative student initialisation or fully matched parameterisation ablation tests this possibility, so the observed ICL difference should be treated as setup-specific rather than attributed to a general preservation mechanism.

6.6.3 Budget-dependent reallocation of computation

Table 6.3 describes the final compressed structures at three budgets. Here, $F = 0.8$ is a higher-retention structural reference, not an additional quality–cost claim or a demonstrated Pareto point. The schedule and compression thresholds were selected through Appendix D.1.3.

Each row summarises the compressed deployment structure at a different budget F . The first three columns count how many of the 42 adapted modules are kept as dense components, approximated by dense-SVD, or dropped to low-rank-only form. The *Avg. rank* column reports the mean effective LoRA rank after gate

F	Dense kept	Dense SVD	Dense dropped	Avg. rank	Speedup vs dense	Param. red.
0.0	0/42	0/42	42/42	128.0	4.05×	80.2%
0.4	9/42	0/42	33/42	127.9	1.74×	53.9%
0.8	24/42	1/42	17/42	122.5	1.15×	30.6%

Table 6.3: Post-training compression statistics for Budgeted LoRA across budgets. Speedups and parameter reduction are measured over the replaced/compressed modules only.

hardening. The final columns translate the resulting structure into compressed-module efficiency.

Results and interpretation. The main structural change is in dense retention: 0, 9, and 24 of 42 dense modules are kept at $F = 0.0$, 0.4 , and 0.8 , respectively; one module enters the dense-SVD case at $F = 0.8$. Average effective rank remains near the maximum (128.0, 127.9, and 122.5), so these runs provide limited evidence of substantial adaptive rank reallocation. The profiles show a budget-dependent change in the final dense/low-rank composition, but, because removal is ordered by cost rather than importance, they do not establish that the retained structure is behaviourally optimal.

6.7 Discussion

In the fixed-student setting studied here, changing the dense-retention budget changes both the final module structure and the measured quality–cost outcome. This extends Part II from paired-checkpoint module transfer to teacher-guided adaptation under a budget, but it does not show that the cost-only controller identifies redundant or behaviourally important modules.

This chapter also clarifies the role of distillation within that story. Portability concerns direct reuse of explicit PEFT modules across hosts. By contrast, the present chapter studies behavioural transfer into a new student parameterisation. Distillation is therefore not a rival notion to portability, but the correct controlled-transfer setting for studying budgeted adaptation when exact module reuse is not

the object of study.

Standard KD LoRA and Budgeted LoRA have similar perplexity at $F = 0.4$ but different mean ICL probe composites at the larger teacher scales. This setting-specific contrast motivates examining behaviour-relevant internal structure in Part III; it does not itself identify a circuit or establish why the probe scores differ.

6.8 Implications for external PEFT modules and internal circuits

6.8.1 Implications for PEFT as reusable adaptation modules

For Part II, the experiments add dense-retention budget and post-training consolidation as design variables alongside fixed-rank LoRA. At $F = 0.4$, the resulting student has similar perplexity to KD LoRA with the same teacher and lower adapted-module proxy costs. This comparison is limited to the tested cost-only rule and does not establish superiority over importance-aware pruning or end-to-end deployment baselines.

Distillation sharpens this conclusion because it functions here as a controlled form of behavioural transfer. When exact module transfer is not the relevant setting, behaviour can still be transferred into a constrained parameter-efficient student parameterisation. The Budgeted LoRA results show that this transfer need not be uniform: under a shared budget, different projections can retain, approximate, or relinquish dense computation in different ways.

6.8.2 Implications for internal circuits and behaviour-relevant structure

The second implication points forward to Part III. In this setup, the budgeted variants have higher mean ICL composite scores than KD LoRA at larger teacher

scales even when perplexity is similar. This correlation suggests a question about how adaptation and compression relate to ICL-relevant structure; it is not a general causal conclusion.

This is a behavioural inference from the probe results rather than a direct circuit-level causal claim; Chapter 7 examines load-bearing internal structure more directly.

The present chapter does not identify which pathways are behaviourally important. Its cost-only controller cannot support that inference, and the copied, frozen student may contribute to the difference between KD LoRA and Budgeted LoRA. The bounded result is a contrast in probe means under one teacher–student construction, which motivates the causal tests in Chapter 7.

6.9 Summary

This chapter addressed the budgeted-adaptation question for Part II of the thesis. Starting from the PEFT-as-module perspective developed in Chapters 4 and 5, it examined how scarce adaptation resources can be organised after behavioural transfer when an explicit dense-compute budget is imposed.

I studied a LoRA-based method with a fixed 6-layer student, module-level dense-retention coefficients, learned rank gates, and post-training compression. Two sampled budgets produce different within-method quality–cost outcomes. At $F = 0.4$, perplexity is close to KD LoRA with the same teacher while the adapted-module proxies are lower; at $F = 0.0$, proxy savings are larger and perplexity is worse. These points are not claimed to be Pareto-optimal, and KD Full—including the 0.13B-teacher case—is the stronger absolute-perplexity baseline.

At larger teacher scales, the budgeted variants also have higher mean scores than KD LoRA on the 19-task function-style ICL composite. This result is specific to the warm-started, frozen-backbone setup and may reflect the altered optimisation parameterisation. Finally, the controller uses cost alone, not module importance; an importance-aware comparison is required before claiming that its allocation is behaviourally well chosen. Chapter 7 therefore moves to direct intervention-based

tests of internal structure.

Part III

Internal Circuits: In-Context Learning and Load-Bearing Structure

Chapter 7

In-Context Learning Capability: Circuit Mechanisms & Load-Bearing Structure

This chapter turns from external PEFT modules to internal circuits. Chapters 4–6 studied reusable behaviour through detachable parameter-efficient fine-tuning (PEFT) components that can be described, transferred, and organised under deployment budgets. The present chapter studies the complementary internal case: reusable computation carried by structures already present inside the model. In particular, the chapter asks what internal structure supports in-context learning (ICL), where a model reuses learned capability at inference time from a prompt, without additional parameter training or weight updates.

This internal case differs from the PEFT-module case in an important way. PEFT modules are architected as explicit detachable components, whereas internal circuits are usually inferred from model behaviour and intervention rather than specified by design. The chapter therefore does not assume that arbitrary circuits satisfy all modularity conditions introduced in Chapter 2. Instead, it operationalises the main internal object more narrowly through attention heads: architecturally defined components with fixed locations, inputs, outputs, and intervention points.

Using *Bi-Induct*, a mechanism-targeted synthetic-data intervention introduced in Section 7.2.1, I test whether candidate ICL-relevant attention-head signatures can be made more visible under matched compute, and whether stronger mechanistic signatures correspond to load-bearing structure.

7.1 Preliminaries

This section establishes the conceptual and operational scaffolding needed to read the rest of the chapter. It builds on the earlier discussion of internal circuits, in-context learning, and load-bearing structure in Chapter 2, Section 2.2.2, together with the internal-circuit methodology introduced in Chapter 3, Section 3.4. The aim here is not to repeat those foundations, but to define the chapter-local terms and measurements needed for the Bi-Induct experiments and the four evaluation axes used later in the chapter.

I first define the operational terms used throughout the analysis, including induction signatures, anti-induction, copy-score telemetry, targeted ablation tests, and load-bearing structure (Section 7.1.1). I then motivate in-context learning (ICL) as a form of reuse without parameter updates (Section 7.1.2), frame the central problem as the distinction between visible mechanistic signatures and load-bearing structure (Section 7.1.3), and position the chapter relative to prior work on induction mechanisms, mechanism-targeted training signals, and matched-compute evaluation (Section 7.1.4). The section closes with the full chapter outline (Section 7.1.5).

7.1.1 Operational definitions: signatures, circuits, and load-bearing tests

This subsection establishes the chapter’s terminology before those terms are used in the motivation, experimental design, and results. It is intentionally placed first because the chapter’s central argument depends on separating three ideas that are easy to conflate: a *signature*, a *candidate circuit*, and a *load-bearing circuit*. Building

on the distinction between structural and functional modularity in Chapter 2, I use *signature* for a measured pattern that makes a component look like a candidate mechanism, and *load-bearing circuit* for internal structure whose causal contribution is tested by intervention. The definitions below introduce the attention-head scores, localisation and spread measures, and ablation quantity used later in the chapter. The broader evaluation protocol is introduced in Section 7.2.4, and supplementary implementation details are provided in Appendix E.2.

Scope of modularity in this chapter. Internal circuits are treated here as candidate reusable structures rather than as modules by assumption. Unlike PEFT modules, their boundaries are usually inferred from behaviour and intervention rather than specified by design. For this reason, the chapter focuses on measurable signatures and targeted ablations: attention heads provide the main operational unit because they are architecturally defined and can be localised, scored, and intervened on directly.

Induction head and induction circuit. Let $S = (s_0, \dots, s_{L-1})$ be a token span of length L , where each s_i is a token, and let $S' = (s'_0, \dots, s'_{L-1})$ be a repeated copy of that span with $s'_i = s_i$. Consider the repeated sequence

$$x = s_0 s_1 \dots s_{L-1} \langle \text{sep} \rangle s'_0 s'_1 \dots s'_{L-1}, \quad s'_i = s_i.$$

An *induction head* is an attention head that supports forward copying across such repeated spans. Operationally, an induction-style head attends from a token s'_i in the second span back to the token that followed the corresponding token s_i in the first span, thereby helping to predict the next token in the repeated sequence. In this chapter, I use the term *induction circuit* for the broader internal computation involving such heads and their interactions with surrounding residual-stream and multi-layer perceptron (MLP) components. The measured attention pattern is therefore a *signature* of the candidate circuit, not by itself proof that behaviour

relies on that circuit.

Anti-induction. *Anti-induction* is the backward-copy counterpart of induction. Rather than retrieving information that helps predict the next token after a repeated cue, anti-induction retrieves information aligned with the previous-token or reversed-span structure. It is included as a directional control: if the synthetic curriculum simply teaches generic equality matching, then backward-copy probes should also become stronger; if transformer copy circuitry is directionally biased, anti-induction scores should remain weak.

Copy-score probes. To detect induction-like and anti-induction-like signatures, I evaluate attention heads on held-out copy probes disjoint from training. These probes are a measurement device, not the Bi-Induct training intervention itself; the training snippets are introduced later in Section 7.2.1. Unless otherwise stated, copy-score probes use long repeated spans with $L = 500$, rather than the $L = 20$ span used in the main Bi-Induct curriculum, to reduce the chance that telemetry simply reflects memorisation of the replacement-based training-intervention format. Appendix E.2.2 gives the supplementary probe-count and concentration details. Let $A^{(\ell,h)} \in \mathbb{R}^{T \times T}$ be the attention map for layer ℓ , head h , with rows indexing target positions and columns indexing source positions. Let t_i denote the position of s'_i in the second span and m_i denote the position of s_i in the first span.

Induction score. The induction score measures attention mass on the next-token alignment:

$$\mathcal{D}_{\text{next}} = \{(t_i, m_{i+1})\}_{i=0}^{L-2},$$

$$\text{Score}_I(\ell, h) = \mathbb{E}_x \left[\frac{1}{L-1} \sum_{(t_i, m_{i+1}) \in \mathcal{D}_{\text{next}}} A_{t_i, m_{i+1}}^{(\ell, h)} \right].$$

Higher Score_I indicates a stronger forward-copy signature.

Anti-induction score. The anti-induction score measures attention mass on the same-token alignment:

$$\mathcal{D}_{\text{same}} = \{(t_i, m_i)\}_{i=0}^{L-1},$$

$$\text{Score}_A(\ell, h) = \mathbb{E}_x \left[\frac{1}{L} \sum_{(t_i, m_i) \in \mathcal{D}_{\text{same}}} A_{t_i, m_i}^{(\ell, h)} \right].$$

Higher Score_A indicates a stronger backward-copy or anti-induction-style signature.

Detecting specialised heads. For each layer ℓ , I rank heads by their copy score and select the top 2% of heads, with a one-head-per-layer floor:

$$k_\ell = \max\{1, \lceil 0.02 |H_\ell| \rceil\}.$$

These selected heads are used as a localisation proxy for the most induction-like or anti-induction-like heads in that layer. This procedure gives a consistent localisation rule across model sizes and curricula. Because the tested models contain 12, 16, or 24 attention heads per layer, the one-head floor applies at every tested scale, so the rule selects the single highest-scoring head in each layer.

The 2% rule was chosen to make the intended localisation sparse and proportional to layer width, while the one-head floor ensures that every layer is represented. In the present models it reduces to exactly one head per layer, so 2% is not empirically distinguished from any other percentage that also rounds to one. The rule is operational rather than uniquely justified, and the telemetry and ablation conclusions are conditional on it; sensitivity to larger top- k sets remains a limitation.

Non-zero signature spread. Because the top-2% localisation rule selects one highest-scoring head per layer for all model sizes studied here, I additionally report the number of selected heads whose measured copy score is strictly greater than zero. Equivalently, this counts the number of layers in which the highest-scoring head exhibits a non-zero measured copy signature. This quantity describes the depth-wise spread of detectable activity; it does not imply that every non-zero score

represents a strongly specialised or behaviourally important head.

Earlier emergence. When I refer to a signature emerging *earlier*, I mean that the strongest copy-score peaks are localised in earlier layers at a fixed final checkpoint under matched compute. I do not mean that the model reaches a fixed behavioural capability in fewer optimisation steps. This distinction matters because the chapter evaluates final checkpoints under iso-FLOPs: the results ask whether targeted data changes the location, visibility, and causal role of copy-style signatures, not whether it provides a full training-time acceleration curve.

Load-bearing induction. A high copy score means that an induction-like signature is detectable, but it does not show that behaviour relies on the corresponding heads for ICL. Following the targeted-ablation methodology introduced in Section 3.4.5, I treat induction structure as *load-bearing* only when intervention shows that behaviour is more sensitive to removing the selected heads than to removing matched random heads. For each model and curriculum, I identify the highest-scoring induction head in every layer under the rule above, ablate those heads, and compare the result with the clean run and a random ablation that removes the same number of heads in the same layers. Matching count and depth controls for the amount and layer distribution of capacity removed, although it does not control for every head property. Implementation details and per-task results are provided in Appendix E.5; the aggregate results are reported in Section 7.3.3.

If $\text{ICL}_{\text{clean}}$ is the clean ICL composite and ICL_{abl} is the score after ablation, I report the relative ablation effect as:

$$\Delta_{\text{abl}} = 100 \times \frac{\text{ICL}_{\text{abl}} - \text{ICL}_{\text{clean}}}{\text{ICL}_{\text{clean}}}. \quad (7.1)$$

A circuit is treated as more load-bearing when targeted induction-head ablation produces a substantially larger negative Δ_{abl} than matched random-head ablation. A large targeted drop therefore indicates behavioural importance beyond the detected

induction signature.

Use in the evaluation. These operational definitions are used most directly in the mechanistic-signature and causal-reliance axes of the evaluation. The induction and anti-induction scores define the copy-score telemetry used to measure **mechanistic signatures**; the top-2% localisation rule and non-zero signature-spread measure describe where and how widely those signatures are detected; the selected induction heads are then used in the targeted-ablation tests; and the relative ablation effect Δ_{abl} defines how I measure **causal reliance**. The remaining two axes are introduced with the full evaluation protocol: **behavioural capability** is measured with few-shot ICL benchmarks, and **language-model quality** is measured with held-out cross-entropy and perplexity. This full four-axis evaluation protocol is specified in Section 7.2.4.

7.1.2 ICL as reuse without parameter updates: why circuits matter

This subsection explains why the definitions above matter for the thesis argument. It first connects ICL to the thesis theme of reuse under constraints, then explains why a data-level intervention is an appropriate way to test whether candidate circuits can be made more visible or more load-bearing. The concrete replacement-based data intervention is not introduced until Section 7.2.1; here the goal is to motivate why such an intervention is worth testing.

A central thesis theme is *efficiency through reusability*. A distinctive form of reuse in large language models (LLMs) is that many tasks can be performed *without* changing parameters: the model is conditioned on a prompt and executes the task via **in-context learning (ICL)**. ICL refers to the ability of a pretrained model to infer a task from *in-context demonstrations* (few-shot examples in the prompt) and execute it at inference time *without updating weights* (Brown et al., 2020). From the thesis perspective, this is **reuse without parameter updates**: prompting works

because inference-time computation routes through internal circuits that implement the required behaviour.

Chapter 6 reported setup-specific differences in mean ICL-probe scores between KD LoRA and Budgeted LoRA within the reported warm-started student setting. The present chapter turns to the complementary internal question: what kinds of circuit structure support ICL behaviour under matched compute, and when do mechanism-like signatures become *load-bearing* for reuse? This motivates an internal-circuit question: if ICL is partly supported by specialised internal structure, can a targeted data intervention make the corresponding signatures more visible, and does that visibility translate into stronger behavioural reliance? In this chapter, the intervention is mechanism-targeted *data*: a minimal synthetic-data intervention designed to preferentially exercise a hypothesised ICL-relevant computation under a fixed compute budget.

I intervene at the *data* level (rather than changing the model or optimiser) for three reasons:

1. **A concrete candidate mechanism exists.** Mechanistic interpretability¹ has identified a simple two-attention-head motif implementing an induction-style copy computation and linked it to few-shot pattern matching (Olsson et al., 2022). This motivates targeting copy-like structure as a plausible internal mechanism supporting reuse without parameter updates.
2. **Under matched compute, data is the main controllable degree of freedom.** Scaling-law analyses relate loss predictably to model size, data, and compute (Kaplan et al., 2020; Hoffmann et al., 2022). More recent work formalises *data quality* as shifting scaling curves and reducing the compute needed to reach a target loss (Bansal et al., 2022; Subramanyam et al., 2026).²

¹Here I use “mechanistic interpretability” in the intervention-based sense: explaining model behaviour via internal components and testing causal contribution with targeted interventions such as ablations or activation patching. The chapter-level operational definitions are given in Section 7.1.1.

²Empirically, some forms of noise/curation primarily shift curves with limited change to exponents in controlled settings (Bansal et al., 2022); quality-aware formulations model this with an explicit quality parameter that trades off with model/data requirements (Subramanyam et al.,

3. Transformer inductive biases favour simple algorithmic primitives.

Transformers tend to learn relatively simple algorithmic behaviours (including copy/search-style primitives), making copy-targeted data interventions a direct way to bias learning toward reusable circuit motifs (Davidovich and Ringel, 2026).

These considerations motivate the core question studied in this chapter:

Can mechanism-targeted synthetic data make ICL-relevant circuit signatures emerge earlier or become more visible, and does this translate into stronger behavioural reliance on the corresponding attention heads under matched compute?

The remainder of the chapter answers this question using the Bi-Induct replacement-based synthetic-sequence intervention (Section 7.2.1).

7.1.3 Problem framing: signatures are not the same as load-bearing structure

This subsection narrows the above motivation into the specific problem tested in the chapter. I first state the tempting hypothesis behind mechanism-targeted pre-training, then explain why it is incomplete, and finally connect the issue back to modularity and matched-compute evaluation. The empirical tests of this problem are deferred to the four evaluation axes in Section 7.3.

A common intuition in mechanism-targeted pretraining is:

If a targeted intervention makes an ICL-relevant circuit appear earlier (or more prominently), then ICL capability should improve.

This sounds plausible because induction heads are causally linked to ICL on controlled tasks and can exhibit phase-transition-like emergence (Chen et al., 2024; Edelman et al., 2024).

2026).

The pitfall. A circuit can be *present* without being *used*. Synthetic cues might increase induction-style *activity* while leaving prediction routed primarily through other pathways. In this thesis, this distinction is captured by **load-bearing structure** (Chapter 2): a component is load-bearing when targeted intervention on that component causes a larger loss in the relevant behaviour than matched control interventions. The chapter measures induction signatures and tests whether the detected structures become *load-bearing* for ICL.

Why this is a modularity question. Mechanism-targeted synthetic data can be viewed as a data-level intervention aimed at a candidate internal circuit signature. Modularity suggests that strengthening a candidate route might make behaviour easier to localise; redundancy suggests the opposite: capability can be implemented along multiple routes, so increasing the visibility of one route does not guarantee that it carries the load. The experiments examine that tension by pairing signature measurements with matched targeted-versus-random ablations in the Bi-Induct setting.

Reuse under constraints. Under a fixed compute budget, any synthetic intervention displaces natural text. Thus, the practical question is: *Does mechanism-targeted synthetic data improve ICL per FLOP under matched compute, without degrading language-model quality?*

7.1.4 Positioning relative to prior work

This subsection locates the chapter within the closest prior work, without repeating the broader literature review in Chapter 2. I organise the positioning around three threads that the experiments combine: mechanistic accounts of ICL and induction, mechanism-targeted training signals, and matched-compute evaluation. The point of this positioning is to make clear why the chapter evaluates Bi-Induct on four axes via behavioural endpoints, telemetry, causal ablations, and a language-model quality check rather than relying on any one of these signals alone.

Mechanisms for ICL and induction. Mechanistic interpretability work has identified induction-style copy circuits and linked them to few-shot pattern matching and ICL-like behaviour in transformers (Olsson et al., 2022). Subsequent work expands the empirical and theoretical picture of induction-head emergence, copy circuits, and the relationship between attention heads and ICL behaviour, motivating copy/induction as a plausible circuit-level mechanism supporting reuse without parameter updates (Chen et al., 2024; Edelman et al., 2024; Veitsman et al., 2025; Yin and Steinhardt, 2025).

Mechanism-targeted training signals. A broader line of work uses targeted examples, curricula, objective tweaks, or data rewrites to encourage specific capabilities (including ICL-relevant behaviours) during training (Gu et al., 2023; Kim et al., 2025; Gloeckle et al., 2024; Maini et al., 2024; Nguyen et al., 2025; Fujii et al., 2025). These works are often evaluated via behavioural outcomes and signature-style diagnostics rather than via explicit tests of *causal reliance* (i.e., whether the induced mechanism becomes load-bearing) (Zhang and Nanda, 2024).

Matched-compute evaluation and the “signature vs. load-bearing” gap. Scaling-law work emphasises matched budgets (Kaplan et al., 2020; Hoffmann et al., 2022), while mechanistic interpretability already places causal intervention at the centre of circuit claims (Zhang and Nanda, 2024). The chapter does not claim causal testing itself as novel. Its narrower contribution is to apply behavioural evaluation, copy telemetry, matched targeted/random ablation, and natural-text perplexity jointly to a matched-compute Bi-Induct case study in which synthetic replacements displace natural text.

Chapter contribution. Bi-Induct operationalises a minimal replacement-based data intervention (forward-copy, backward-copy control, and balanced variants) and evaluates it under iso-FLOPs across model scales. The chapter uses this intervention as a controlled case study, not as a claim that Bi-Induct is a generally superior

pretraining recipe. Within this case study, the contribution is the joint application of capability endpoints, copy-head telemetry, matched targeted-versus-random ablations, and a natural-text quality check to separate *induction signatures* from *load-bearing induction*.

7.1.5 Chapter outline

Having defined the main terms and motivated the problem, the remainder of the chapter proceeds as follows. Section 7.2 introduces the experimental contribution: the Bi-Induct data intervention, the model family, the matched-compute setup, and the four-axis evaluation protocol. Section 7.3 reports the four corresponding result axes: behavioural capability, mechanistic signatures, causal reliance, and language-model quality. Section 7.4 then discusses the implications for mechanism-targeted synthetic data and for the broader thesis narrative of modularity for reuse under constraints. Section 7.5 concludes.

7.2 Experiments

This section describes the experimental contribution of the chapter. It first introduces Bi-Induct, the mechanism-targeted replacement-based data intervention used to test whether copy-style circuits can be strengthened under matched compute (Section 7.2.1). It then states the experimental scaffold (Section 7.2.2), gives the concrete model family, data, training, and matched-compute details (Section 7.2.3), and defines the four evaluation axes used to interpret the results (Section 7.2.4). The purpose of this section is therefore to make clear what is being changed, what is being held fixed, and how the resulting models are evaluated before moving to the results in Section 7.3.

7.2.1 Bi-Induct: a minimal mechanism-targeted data intervention

This subsection introduces **Bi-Induct**, the mechanism-targeted data intervention used in this chapter. Bi-Induct is not proposed as a generally superior pretraining recipe. Its role is experimental: it adds controlled copy-structured snippets to an otherwise natural pretraining stream and tests whether induction-like circuitry becomes more visible and more load-bearing under matched compute. In the terminology of Section 7.1.1, Bi-Induct targets mechanistic *signatures*; the later evaluation tests behavioural capability and causal reliance separately.

Bi-Induct is implemented as a *replacement-based data intervention*. During streaming pretraining, each candidate natural-sequence slot is either retained or replaced by a short synthetic directional-copy snippet. The synthetic snippet is padded to the fixed context length, and padded positions are excluded from the language-modelling loss. It therefore occupies the same fixed-shape batch slot as the displaced natural sequence, but contributes loss only on its non-padding tokens. Holding batch shape, sequence length, and optimiser steps fixed preserves nominal FLOPs across curricula, while each replacement displaces natural-text exposure and typically reduces the number of non-padding loss-bearing tokens. The experiment therefore evaluates the complete replacement-based intervention under matched compute, rather than isolating copy structure at a matched number of supervised tokens.

The intervention is based on the induction-head motif studied in prior mechanistic work on in-context learning: when a token or span reappears in context, an induction-style computation retrieves what followed the earlier occurrence and uses it to predict what should follow the later occurrence (Olsson et al., 2022). In this chapter, *induction* denotes this forward-copy direction. *Anti-induction* denotes the backward-copy control: instead of repeating the span in the same order, the second half reverses it. This control tests whether the model learns a generic equality/repetition cue or whether transformer copy circuitry remains directionally biased

toward forward induction.

Let $\mathcal{V}$ be the tokenizer vocabulary and let $\text{BPE}(\cdot)$ denote the byte-pair encoding (BPE) tokenizer. For a span length L , I sample a token span

$$S = (s_1, \dots, s_L), \quad s_i \sim \text{Uniform}(\{\lfloor 0.05|\mathcal{V}| \rfloor, \dots, \lfloor 0.95|\mathcal{V}| \rfloor\}).$$

This middle-range sampling excludes the lowest and highest token IDs, which in the tokenizer include special, reserved, or otherwise atypical IDs. The purpose is not to sample according to natural token frequency, but to avoid snippets dominated by special symbols or rare tokenisation artefacts. I use a single space as a neutral separator, $\text{SEP} = \text{BPE}(" ")$.

Random token spans were chosen to isolate equality, order, and directional copying from semantic predictability. This makes the copy relation controlled, but the snippets are less natural and less useful as language data than real-text spans; the study does not compare the two designs. Standard next-token loss is applied to all non-padding tokens in both spans. Consequently, most of the first random span is unpredictable from its prefix and contributes noisy or irreducible loss, whereas the second span is predictable from the earlier copy. Masking loss on the first span would still allow second-span gradients to flow through its contextual representations and would provide a cleaner test of the copy objective. Because that ablation was not run, the present behavioural and perplexity results conflate the copy intervention with this first-span loss choice.

Forward/induction snippets (Figure 7.1, top). The forward-copy condition repeats the sampled span in the same order:

$$\text{Induction}(S) = [S \parallel \text{SEP} \parallel S].$$

This directly exercises the copy pattern associated with induction heads: the second occurrence of the span provides a cue to retrieve what followed the corresponding

position in the first occurrence.

Backward/anti-induction snippets (Figure 7.1, bottom). The backward-copy condition reverses the second span:

$$\text{Anti}(S) = [S \parallel \text{SEP} \parallel \text{reverse}(S)].$$

This provides a directional control. If the replacement data simply strengthens generic matching over repeated tokens, anti-induction scores should also increase. If the model’s copy circuitry is biased toward the forward induction direction, anti-induction should remain weak.

Balanced snippets. The balanced condition mixes the two directions. On each synthetic replacement, I flip a fair coin to choose between an induction snippet and an anti-induction snippet. Each snippet has length $\ell_{\text{snip}} = 2L + |\text{SEP}|$; for example, if SEP is a single space token, the snippet length is $2L+1$.

Scheduling the Bi-Induct replacement under iso-FLOPs. The Bi-Induct replacement schedule is front-loaded and annealed. Let m_0 be the initial mix ratio, defined as the initial probability that a candidate natural-sequence slot is replaced by a synthetic snippet. Let t denote the natural-token progress used by the data iterator to drive the schedule, and let T_a be the annealing horizon in the same units. The instantaneous replacement probability is

$$m(t) = \max\{m_0 \cdot (1 - t/T_a), 0\}.$$

For each candidate natural-sequence slot, I draw $u \sim \text{Uniform}(0, 1)$. If $u < m(t)$, the natural sequence is replaced by one synthetic snippet according to the current `synthetic_task` $\in \{\text{induction}, \text{anti}, \text{balanced}\}$. Otherwise, the natural sequence is retained. Loss masking and fixed-shape batching follow the convention defined at the start of this subsection.

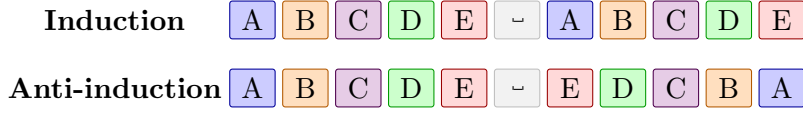


Figure 7.1: Examples of copy-style snippets used in the Bi-Induct replacement-based data intervention. Each snippet is a span of L random non-special tokens, followed by a separator, then either the same span (induction) or the reversed span (anti-induction). Colours align repeated tokens across the two halves. The illustration uses $L=5$ for clarity.

Expected average replacement probability ($\bar{m}$). Because replacement is determined by a Bernoulli draw for each candidate sequence slot, the realised number of replacements varies slightly across runs. I summarise the schedule using its expected average replacement probability over the baseline training horizon T_{base} :

$$\bar{m} = \frac{1}{T_{\text{base}}} \int_0^{T_{\text{base}}} m(t) dt = \begin{cases} m_0 \left(1 - \frac{T_{\text{base}}}{2T_a}\right), & T_{\text{base}} \leq T_a, \\ m_0 \frac{T_a}{2T_{\text{base}}}, & T_{\text{base}} > T_a. \end{cases}$$

For the main experiments, $T_a = T_{\text{base}}$, so $\bar{m} = m_0/2$. This quantity summarises the expected replacement probability over training progress; it is not the realised fraction of replaced slots or the fraction of non-padding training tokens that are synthetic.

Why this schedule? The schedule has three purposes. First, it front-loads the mechanism-targeted signal and anneals it away to reduce late-stage distributional shift. Second, it exposes a single intervention-strength dial, m_0 . Third, it retains the fixed-shape matched-compute comparison defined above. The runs are matched in nominal FLOPs but not in non-padding loss-bearing tokens or natural-text exposure.

7.2.2 Experimental setup and measurement

This subsection explains how the Bi-Induct experiments instantiate the shared internal-circuit methodology introduced in Chapter 3. In particular, the experiments use the same methodological principles as Section 3.4: matched-compute

comparison, mechanism-aware telemetry, targeted intervention tests for load-bearing structure, and a separate check on general language-model quality. The chapter-local operational definitions for signatures, top-2% head localisation, and targeted-versus-random ablation are given in Section 7.1.1. The concrete model, data, and training setup is given in Section 7.2.3; the reported measures are then organised into the four-axis evaluation protocol in Section 7.2.4.

7.2.3 Models, data, and matched compute

I train decoder-only Transformers at 0.13B, 0.5B, and 1B parameters. The natural-only baselines use the same model family, corpus, and preprocessing pipeline as Section 6.4.1. The models are trained on deduplicated THE PILE with a fixed tokenizer and stable held-out split (Gao et al., 2020); sequence shape and loss masking are specified once in Section 7.2.1. The three intervention conditions are Induction, Anti-Induction, and Balanced.

The model family follows the decoder-only Transformer presets reported in Table 6.1 and introduced in Section 6.4.1 of Chapter 6. Each model uses pre-norm residual blocks, rotary position embeddings (RoPE), grouped-query attention (GQA), SwiGLU gated feed-forward layers, head dimension 64, and untied input/output embeddings. The 0.13B, 0.5B, and 1B presets inherit the layer counts, hidden sizes, attention-head counts, and key-value head counts reported in Table 6.1.

For language-model quality evaluation, I reserve a fixed 0.2% held-out slice of THE PILE (approximately 0.4B tokens), selected by stable hash partition. All runs use a context length of 1,024 tokens, and results are averaged over three random seeds. Training uses bfloat16 precision and AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay 0.1, peak learning rate 10^{-3} , 3% linear warmup, and cosine decay. Each update processes 2^{16} nominal token positions under the fixed-shape and loss-masking convention defined in Section 7.2.1. Following compute-optimal guidance (Hoffmann et al., 2022), for N non-embedding parameters I set $T_{\text{base}} \approx 20N$, compute $U = \lceil T_{\text{base}}/2^{16} \rceil$, and keep U fixed across curricula at each scale. This matches nom-

inal FLOPs, batch shape, and optimiser steps, but not non-padding loss-bearing tokens or natural-text exposure; the comparison therefore evaluates the complete replacement intervention rather than a token-matched copy objective.

I use the 0.13B model as a design lab to choose the Bi-Induct operating point before scaling to the 0.5B and 1B models. This means that I first sweep intervention hyperparameters at the smallest scale, where repeated runs are cheapest, and then fix the selected values for the larger-scale comparison. The span-length sweep tests $L \in \{5, 20, 500\}$ with a fixed initial mix ratio of 25%; $L = 20$ is selected because it provides a stable middle point: very short spans provide a weaker copy signal, while very long spans do not give a consistent ICL gain and can slightly worsen perplexity. The mix-ratio sweep then fixes $L = 20$ and tests $m_0 \in \{25\%, 50\%, 100\%\}$; $m_0 = 50\%$ is selected as a moderate intervention strength that gives a clearer mechanism-targeted signal than 25% while avoiding the much larger language-model quality degradation observed at 100%. The full span-length and mix-ratio sweeps are reported in Appendices E.3.1 and E.3.2. The scaling experiments therefore use $L = 20$ and $m_0 = 50\%$, linearly annealed to zero over the full training budget.

7.2.4 Four-axis evaluation: behavioural capability, mechanistic signatures, causal reliance, and language-model quality

Bi-Induct is evaluated using four axes: behavioural capability, mechanistic signatures, causal reliance, and language-model quality. The first three axes test whether the relevant behaviour is present, whether the intended internal signature is detectable, and whether behaviour relies on the detected structure. The fourth axis separately evaluates whether the replacement-based intervention preserves general language-model quality on natural text.

1. **Behavioural capability.** I evaluate two few-shot ICL families. The first is a standard language-model benchmark family evaluated in a 3-shot set-

ting. It includes MMLU, Winogrande, CommonSenseQA, PIQA, HellaSwag, TriviaQA-Wiki, BBH, OpenBookQA, ARC-Challenge, GPQA, GSM-8K, MathQA, BoolQ, and LAMBADA. Multiple-choice and classification tasks are evaluated with accuracy; open-domain question-answering, maths, and generation-style tasks are evaluated with exact match where that is the standard task metric. The second family is the function-style ICL probe suite of Todd et al. (2024), evaluated in a 10-shot setting with HITS@1 accuracy. These probes test simple string-manipulation and selection behaviours that are more directly sensitive to in-context copying and selection than broad knowledge benchmarks. In both families, I report macro-averaged composites over tasks. The full benchmark list, per-task metrics, and shot counts are given in Appendix E.2.1, Table E.1; full per-task results are reported in Appendix E.4.1.

2. **Mechanistic signatures.** I compute per-head induction and anti-induction scores on held-out copy probes disjoint from training. These scores measure whether copy-style signatures are detectable and where they are localised across model depth. The score definitions and top-2% head-localisation rule are given in Section 7.1.1; supplementary telemetry details are provided in Appendix E.2.2.
3. **Causal reliance.** I test whether the detected induction heads are behaviourally relied upon by comparing targeted induction-head ablation with matched random-head ablation. The reported quantity is the relative change in the function-probe ICL composite after ablation, as defined in Equation 7.1. This axis tests whether the heads identified by telemetry are merely visible or whether ICL performance depends on them. The main causal-reliance results are reported in Section 7.3.3; supplementary per-task ablation tables are provided in Appendix E.5.
4. **Language-model quality.** I report next-token cross-entropy and perplexity (PPL) on the fixed held-out THE PILE validation slice. For a token sequence

$x_{1:T}$, the per-token cross-entropy is

$$\mathcal{L}_{\text{CE}}(x_{1:T}) = -\frac{1}{T} \sum_{t=1}^T \log p_{\theta}(x_t \mid x_{<t}),$$

and perplexity is

$$\text{PPL} = \exp(\mathcal{L}_{\text{CE}}).$$

Under iso-FLOPs comparisons, this quality axis checks whether a mechanism-targeted data intervention improves or amplifies ICL-relevant behaviour at the cost of worse general language modelling on natural text.

This four-axis design is intended to separate questions that would otherwise be conflated. It allows the results to test whether Bi-Induct changes behavioural ICL performance, whether it changes the visibility of induction-like signatures, whether the detected heads are causally relied upon, and whether the intervention preserves language-model quality on natural text.

7.3 Results

This section reports the empirical findings in the same order as the four-axis evaluation protocol in Section 7.2.4. Each subsection first restates what is tested and how the reported output is produced, then describes the observed patterns, and only then interprets those patterns in relation to the chapter’s research question. Section 7.3.1 evaluates behavioural ICL capability; Section 7.3.2 examines induction-like and anti-induction-like signatures; Section 7.3.3 tests whether the detected induction heads are behaviourally relied upon; and Section 7.3.4 assesses the effect of the data intervention on general language-model quality.

7.3.1 Evaluation axis 1: Behavioural capability under matched compute

This subsection evaluates whether Bi-Induct changes the model’s ability to infer and perform tasks from demonstrations in the prompt, without additional parameter updates. I report results for two task families because they test complementary forms of ICL behaviour: broad language-model benchmarks and targeted function-style probes.

What is tested and how. The first family contains 14 standard language-model benchmarks evaluated with three in-context demonstrations. These tasks cover knowledge and reasoning, such as MMLU, GPQA, ARC-Challenge, and GSM-8K; commonsense and continuation, such as Winogrande, PIQA, and HellaSwag; and reading, question answering, and cloze completion, such as BoolQ, TriviaQA-Wiki, and LAMBADA. For multiple-choice or classification tasks, the selected answer is evaluated using accuracy. For generated-answer tasks, including TriviaQA-Wiki, BBH, and GSM-8K, the generated answer is evaluated using exact match. The standard-benchmark composite is the unweighted macro-average of the 14 task scores.

The second family contains 19 function-style probes adapted from Todd et al. (2024), evaluated with ten in-context demonstrations. These probes require the model to infer a simple transformation or selection rule from demonstrations and apply it to a held-out query. Representative tasks include `capitalize`, which requires changing the case of an input; `choose_first_of_3`, which requires selecting the first item from a presented set; `next_item`, which requires identifying the next element under the demonstrated relation; and `word_length`, which requires returning the length of an input word. For each query, HITS@1 is one when the target answer is the model’s highest-ranked output and zero otherwise. The function-probe composite is the unweighted macro-average across the 19 probes.

For both families, prompt templates and shot counts are fixed across curricula,

and results are averaged over three model-training seeds. Figure 7.2 reports the two macro composites; the full task inventory, metrics, and per-task results are provided in Appendix E.2.1, Table E.1, and Appendix E.4.1, Table E.4.

Observed results. On the standard language-model benchmark composite, the differences between the natural-only Baseline and the Bi-Induct curricula are small at all three model scales. At 0.13B, the Baseline scores 22.7 ± 0.5 , while the closest Bi-Induct condition, Anti-Induction, scores 22.5 ± 0.4 . At 0.5B, Induction scores 23.9 ± 0.4 , compared with 23.6 ± 0.8 for the Baseline. At 1B, Balanced and Baseline score 24.3 ± 0.3 and 24.2 ± 0.5 , respectively. Thus, no Bi-Induct curriculum produces a consistent cross-scale advantage on this task family.

The function-style probes show a different pattern. At 0.13B and 0.5B, differences between curricula are mixed: the Baseline scores 5.3 ± 0.9 and 9.6 ± 1.1 , while the best Bi-Induct condition scores 5.2 ± 0.8 and 10.4 ± 1.1 , respectively. At 1B, however, the Baseline reaches 20.0 ± 1.3 , compared with 15.2 ± 1.1 for Induction, 14.7 ± 1.2 for Anti-Induction, and 14.9 ± 1.1 for Balanced.

The 1B Baseline advantage is also visible on individual probes that directly require transformation or selection. For example, on `capitalize`, the Baseline scores 54.8 ± 2.1 , compared with 33.4 ± 1.3 for Induction. On `choose_first_of_3`, the corresponding scores are 69.4 ± 1.8 and 52.4 ± 2.0 ; on `choose_first_of_5`, they are 55.5 ± 1.3 and 42.6 ± 3.0 .

Interpretation and finding. The standard-benchmark results show that replacing part of the natural-text exposure with directional copy snippets does not produce a general few-shot improvement under matched compute. The function-probe results further show that directly exercising a copy-related mechanism during pretraining does not guarantee stronger performance on tasks that appear closely related to copying and selection. At the largest tested scale, the natural-only Baseline performs substantially better on this targeted task family.

The behavioural finding is therefore that Bi-Induct is broadly neutral on the

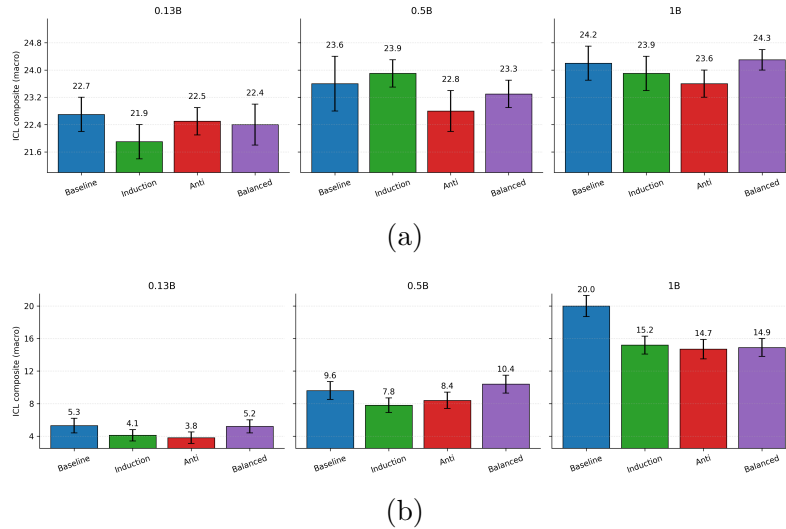


Figure 7.2: ICL composite (macro) across two evaluation families: (a) Standard LM benchmarks; (b) Todd et al. (2024)’s function-probe suite. Each panel groups by model size (0.13B, 0.5B, 1B), with bar colours identifying the training regime (Baseline, Induction, Anti, Balanced); error bars show ± 1 standard deviation. For per-task results see Appendix E.4.1, Table E.4.

standard benchmark composite and can underperform natural-only training on targeted function-style ICL probes. Whether this behavioural difference corresponds to differences in the visibility or causal role of induction-like circuitry is examined in the next two evaluation axes.

Robustness and supplementary results. The robustness experiments in Appendix E.4.2 change the main evaluation from three-shot standard benchmarks and ten-shot function probes to a unified one-shot setting. The standard-benchmark composite changes little across model scales, whereas the function-probe composite decreases more clearly at 0.5B and 1B. Label-permutation and HITS@3 stress tests similarly change absolute scores but preserve the broad ordering between the Baseline and Bi-Induct curricula.

7.3.2 Evaluation axis 2: Mechanistic signatures through induction and anti-induction telemetry

This subsection tests whether Bi-Induct changes the presence, strength, and depth-localisation of induction-like and anti-induction-like attention patterns. Unlike the

behavioural axis, this analysis does not evaluate generated task answers; it measures internal attention patterns at the final checkpoint.

What is tested and how. For each model and curriculum, I evaluate attention heads on 50,000 fresh copy probes that are disjoint from pretraining. Each probe has the form $s \langle \text{sep} \rangle s$, where s is a uniformly sampled token span of length $L = 500$. This is substantially longer than the $L = 20$ snippets used in Bi-Induct training, reducing the likelihood that the measurement merely detects memorisation of the replacement-snippet format.

For every attention head, the induction score measures attention mass on the forward next-token alignment, while the anti-induction score measures attention mass on the backward or same-token alignment, as defined in Section 7.1.1. For each layer, Figure 7.3 plots the best-scoring head selected under the top-2% localisation rule, averaged over three seeds. The top row reports induction scores; the bottom row reports anti-induction scores; and the three columns correspond to the 0.13B, 0.5B, and 1B models.

Observed results. At 0.13B and 0.5B, several Bi-Induct conditions produce prominent induction-score peaks at earlier layers than the natural-only Baseline. The shift is approximately three layers at 0.13B and two layers at 0.5B. At 1B, this ordering reverses: the Baseline produces the earliest clear induction peaks, around layers 10–11, before the Bi-Induct variants.

Peak strength also changes with scale. At 0.13B and 0.5B, the highest measured induction scores approach 1.0, whereas at 1B the strongest peaks remain below 0.5. The depth-wise spread of non-zero measured induction activity also differs across scales. Because the localisation rule selects one highest-scoring head per layer, these counts indicate the number of layers in which the selected head has a measured induction score greater than zero. At 0.13B, the Baseline has non-zero selected heads in three layers, compared with five for Balanced; at 0.5B, the corresponding counts are seven and eight; and at 1B, the Baseline has twelve compared with six

for Induction. Thus, Bi-Induct produces slightly more widely distributed non-zero activity at the two smaller scales, whereas the 1B Baseline exhibits the wider depth-wise spread.

Anti-induction scores remain substantially smaller than induction scores across all scales and curricula. Even the Anti-Induction curriculum does not produce a strong backward-copy signature. The largest observed anti-induction peak is approximately 0.04 at 0.5B, while peaks at 1B remain approximately 0.02 or lower.

Interpretation and finding. The induction-score results show that Bi-Induct can alter where and how visibly induction-like attention patterns appear, particularly at the two smaller scales. However, this effect is not consistent across scale: at 1B, the natural-only Baseline forms the earlier and broader induction pattern.

The near-zero anti-induction scores show that exposure to backward-copy snippets does not symmetrically produce backward-copy attention signatures. This supports a directional asymmetry between forward induction and the anti-induction control.

These results establish that Bi-Induct can change mechanistic telemetry. They do not establish whether the detected heads are required for ICL behaviour. The targeted-ablation analysis in Section 7.3.3 therefore tests whether the visible induction-like structure is also load-bearing.

7.3.3 Evaluation axis 3: Causal reliance through load-bearing head ablations

This subsection tests whether ICL performance depends on the induction heads identified by the telemetry analysis. The intervention is evaluated on the function-style probes because these tasks directly exercise transformations, copying, and selection and therefore provide a more targeted behavioural endpoint than the broader standard-benchmark composite.

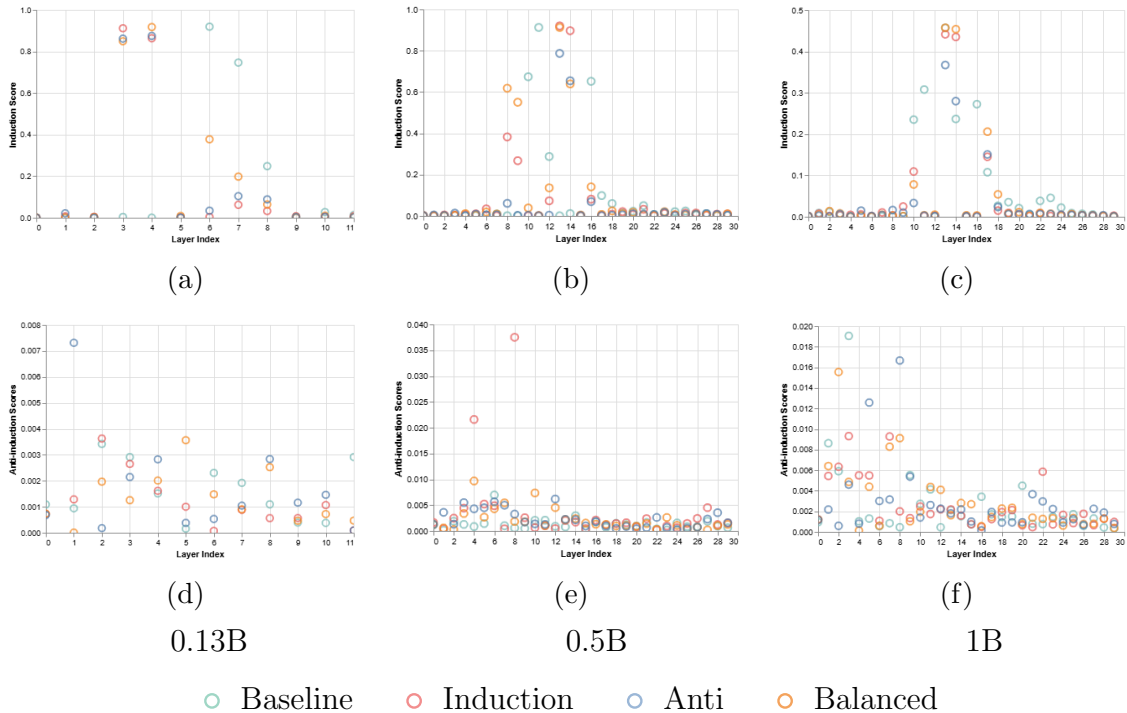


Figure 7.3: Layer-wise copy-head telemetry. Top row: induction scores; bottom row: anti-induction scores. For each layer I plot the best-scoring head (top 2% by score with a floor of one head per layer), averaged over three seeds, for the 0.13B, 0.5B, and 1B models. For head counts for each model see Table 6.1 (Chapter 6).

What is tested and how. For each model and curriculum, I select the top-2% highest-scoring induction heads in every layer, using the one-head-per-layer floor defined in Section 7.1.1. During evaluation, I ablate the selected heads by zeroing their post-attention value-stream outputs before the layer output projection, while leaving queries, keys, and attention weights unchanged. I then evaluate the same ten-shot function-probe prompts under three conditions:

1. the clean model, with no heads ablated;
2. targeted induction-head ablation; and
3. matched random-head ablation, which removes the same number of randomly selected heads per layer.

For each condition, I compute the macro-average of the 19 function-probe HITS@1 scores. Table 7.1 reports the percentage change relative to the corresponding clean

model. Negative values indicate lower performance after ablation. The targeted-versus-random comparison is central: a large targeted drop is evidence of load-bearing induction structure only when it exceeds the effect of removing a matched number of arbitrary heads. Full value-stream-zeroing details and per-task ablation results are provided in Appendix E.5.

Observed results. For the natural-only Baseline, targeted induction-head ablation reduces the function-probe composite at every scale: by 22.6% at 0.13B, 14.6% at 0.5B, and 19.5% at 1B. The matched random-head changes are +17.0%, -4.2%, and -4.0%, respectively. The targeted ablation therefore produces a substantially worse outcome than the matched random control at each scale.

The Bi-Induct curricula generally show smaller targeted-ablation drops. At 0.13B, the targeted drops are 4.9% for Induction, 5.3% for Anti-Induction, and 19.2% for Balanced. At 0.5B, they are 10.3%, 8.3%, and 12.5%, respectively. At 1B, they are 14.5%, 12.9%, and 8.7%. In particular, the difference between the Baseline and the Bi-Induct conditions is largest at 1B, where the Baseline has both the highest clean function-probe composite and the largest targeted-ablation drop.

Some matched random-head ablations produce positive changes, most notably the +17.0% result for the 0.13B Baseline. These values mean that removing an arbitrary set of heads can occasionally improve the aggregate probe score. They should not be interpreted as evidence that random ablation is beneficial; rather, they make the matched targeted-versus-random comparison necessary and show that absolute ablation changes can be sensitive to which heads are removed.

Interpretation and finding. The observed pattern does not match the expectation that stronger or more visible induction telemetry should necessarily correspond to greater behavioural dependence. Although Bi-Induct can make induction-like signatures more visible, the natural-only Baseline is generally more sensitive to removing its highest-scoring induction heads.

Under this head-selection and value-stream-ablation protocol, the natural-only

Model	Baseline		Induction		Anti		Balanced	
	Δ_{induct}	Δ_{rand}	Δ_{induct}	Δ_{rand}	Δ_{induct}	Δ_{rand}	Δ_{induct}	Δ_{rand}
0.13B	-22.6%	+17.0%	-4.9%	+7.3%	-5.3%	+5.3%	-19.2%	0.0%
0.5B	-14.6%	-4.2%	-10.3%	+3.8%	-8.3%	-1.2%	-12.5%	0.0%
1B	-19.5%	-4.0%	-14.5%	-2.6%	-12.9%	+0.7%	-8.7%	-4.0%

Table 7.1: Percent change in function-probe ICL composite after ablating either top-2% induction heads per layer (Δ_{induct}) or a matched number of random heads (Δ_{rand}), relative to the clean run. Negative is worse.

Baseline is more sensitive to targeted removal than the Bi-Induct variants in most comparisons. Bi-Induct can therefore amplify or redistribute the measured signatures without consistently increasing reliance on the selected heads. This conclusion is conditional on the top-one-per-layer localisation rule and the tested function probes.

7.3.4 Evaluation axis 4: Language-model quality check

This subsection evaluates whether Bi-Induct changes general next-token modelling quality on natural text. This axis is necessary because the synthetic snippets occupy part of the fixed training budget: an intervention that changes ICL-related behaviour or telemetry may still be undesirable if it substantially worsens modelling of the natural-data distribution.

What is tested and how. I evaluate each final checkpoint on the same fixed 0.2% held-out slice of THE PILE, selected by stable hash partition and containing approximately 0.4B natural-text tokens. No synthetic snippets are included in this evaluation. Using the same tokenizer and context length for all models, I compute the average next-token cross-entropy over the held-out tokens and convert it to perplexity using the definition in Section 7.2.4. Lower perplexity indicates better prediction of the natural-text sequence. Table 7.2 reports the mean across three seeds.

Curriculum	0.13B	0.5B	1B
Induction	25.8	17.9	14.9
Anti-induction	26.2	18.2	14.9
Balanced	26.2	18.2	14.9
Baseline	21.8	16.0	14.1

Table 7.2: Held-out perplexity (PPL $\downarrow$) on the fixed THE PILE eval split at iso-FLOPs. Values are averaged over three seeds. For each model size, synthetic curricula use a **50%** initial mix ratio linearly annealed over the full training budget.

Observed results. The natural-only Baseline has the lowest held-out perplexity at every scale. At 0.13B, the Baseline perplexity is 21.8, compared with 25.8 for Induction and 26.2 for both Anti-Induction and Balanced. At 0.5B, the Baseline scores 16.0, compared with 17.9–18.2 for the Bi-Induct curricula. At 1B, the Baseline scores 14.1, while all three Bi-Induct curricula score 14.9.

The absolute gap between natural-only and Bi-Induct training decreases with scale: approximately 4.0–4.4 perplexity points at 0.13B, 1.9–2.2 points at 0.5B, and 0.8 points at 1B.

Interpretation and finding. Bi-Induct consistently incurs a natural-text modelling penalty under matched compute: every Bi-Induct curriculum produces higher held-out perplexity than the natural-only Baseline at each tested scale. The degradation is nevertheless limited rather than catastrophic. Relative to the Baseline, the perplexity increase is approximately 18%–20% at 0.13B, 12%–14% at 0.5B, and 6% at 1B. The Bi-Induct models therefore retain substantial natural-text modelling capability, although they do not match the quality of natural-only training.

The experiment does not isolate why the relative penalty decreases with model size. Possible contributing factors include general scaling effects, differences in sensitivity to the replacement-based intervention, and the reduced natural-text and non-padding loss-bearing-token exposure under Bi-Induct. The language-model-quality finding is therefore that Bi-Induct introduces a measurable but non-catastrophic quality trade-off under the tested conditions.

7.4 Discussion

This section interprets the four result axes together. Section 7.4.1 states the main chapter-level implication: reuse without parameter updates requires load-bearing structure; visible signatures alone are insufficient. Section 7.4.3 then applies the four-axis evaluation protocol to mechanism-targeted synthetic data. Section 7.4.4 connects the findings back to the thesis narrative of modularity for reuse under constraints.

7.4.1 Implications: reuse without parameter updates requires load-bearing structure

Across the tested settings, Bi-Induct alters the visibility and localisation of induction-like signatures without a consistent few-shot ICL gain. The targeted/random ablation comparison also does not show increased reliance on the selected heads. Salient copy telemetry is therefore insufficient evidence of behavioural dependence in this case study.

This interpretation follows the established mechanistic-interpretability requirement that circuit claims need causal intervention, not signature visibility alone. For the thesis, the targeted-versus-random ablation supplies that check for one candidate attention-head set; it does not introduce causality as a new criterion for the field.

7.4.2 Limitations of the Bi-Induct design and causal test

Four limitations bound the result. First, matching fixed batch shape and optimiser steps matches nominal FLOPs but not non-padding supervised tokens or natural-text exposure. Second, the random-token construction isolates copying from semantics, but language-model loss is also applied to the unpredictable first span; a masked-first-span or real-text-copy comparison could separate the copy signal from this noisy objective. Third, the top-2% rule collapses to one head per layer at every

tested scale, and no larger top- k sensitivity analysis is reported. Fourth, the causal test covers value-stream zeroing on selected attention heads and one function-probe family; other circuit components, intervention sites, and behavioural endpoints may give a different account. The conclusions are therefore specific to three model scales, three seeds, the stated replacement schedule, and this localisation/ablation protocol.

7.4.3 Applying the four-axis protocol to mechanism-targeted synthetic-data interventions

The four-axis protocol used here applies an established causal principle to the matched-compute data-intervention setting: telemetry alone does not establish a functional circuit. For this case study I therefore report four complementary checks rather than presenting signature visibility as the success criterion:

1. **Measure behavioural capability.** Determine whether the intervention improves or preserves meaningful task endpoints under matched compute.
2. **Measure mechanistic signatures.** Use telemetry to test whether the targeted structure becomes more visible, stronger, or differently localised.
3. **Measure causal reliance.** Because telemetry is correlational, use interventions such as ablation, activation patching, or mediation analysis to test whether behaviour depends on the detected structure.
4. **Measure language-model quality.** Evaluate held-out cross-entropy, perplexity (PPL), and related quality measures to determine whether the intervention degrades modelling of the natural-data distribution.

Design implication for follow-up experiments. Repeatedly presenting a pattern may increase its telemetry without making it instrumental when alternative pathways remain available. Follow-up interventions should therefore test objectives that require the targeted computation, together with matched causal controls and

the token/loss controls listed above. This is a study-specific design implication, not a claim that prior mechanistic-interpretability work neglected causality.

7.4.4 Connection to modularity and the broader thesis

From the thesis perspective, Bi-Induct is a data-level case study aimed at a candidate internal circuit under a fixed nominal compute budget. In this setting, increased signature salience is not accompanied by improved ICL or greater sensitivity to targeted ablation. The result therefore supplies a negative example for the thesis distinction between a detectable structure and a functionally relied-upon one.

Chapter 6 reported setup-specific differences in an ICL probe composite after alternative adaptation procedures. The present chapter uses a separate matched-compute intervention and targeted/random ablation to ask whether visible copy-related heads carry behaviour. Together, they motivate the more limited synthesis in the conclusion: external cost organisation and internal causal reliance are distinct empirical questions and require different controls.

7.5 Chapter summary

This chapter studied internal circuits as possible mechanisms supporting reuse without additional parameter updates through in-context learning (ICL). Using the Bi-Induct mechanism-targeted replacement-based intervention, I tested whether replacing a controlled fraction of natural sequence slots with directional copy snippets during pretraining changes ICL capability, induction-related circuit signatures, behavioural reliance on the detected heads, and general language-model quality under matched compute. The four evaluation axes support the following findings:

- Bi-Induct changes the measured induction-head *signatures* and, at the smaller model scales, shifts some peaks towards earlier layers. It does not provide a consistent few-shot ICL improvement under the fixed-shape comparison.

-
- Anti-induction scores remain near zero even when training includes explicit backward-copy snippets, revealing a strong directional asymmetry in the copy-related circuitry learned by the tested models.
 - Under the top-one-per-layer selection and value-stream-ablation protocol, targeted ablations generally produce larger reductions for the natural-only Baseline than for the Bi-Induct variants. This is evidence of greater reliance on those selected heads within the tested function probes, not a complete circuit identification.
 - Bi-Induct consistently worsens held-out natural-text modelling quality relative to natural-only training, producing higher perplexity at every tested scale. However, the degradation is limited rather than catastrophic: the relative perplexity penalty decreases from approximately 18%–20% at 0.13B to approximately 6% at 1B. Thus, Bi-Induct retains substantial language-modelling capability, but does not preserve the quality achieved by natural-only training under the same nominal compute budget.

The bounded conclusion is that this Bi-Induct implementation can make candidate signatures more visible without increasing measured reliance on the selected heads. It reinforces, rather than originates, the need for causal tests when relating mechanistic signatures to behaviour and provides the thesis’s internal-circuit case study for Chapter 8.

Chapter 8

Conclusion

8.1 Thesis summary

This thesis studied how large language models (LLMs) can be made more reusable and efficient by organising capability under explicit constraints. Modern foundation models are powerful, but costly to train, adapt, and deploy when treated as monolithic systems. I therefore use *modularity* as a practical mechanism for *reuse under constraints*: a way of preserving, transferring, and selectively invoking useful behaviour under limits such as parameters, floating-point operations (FLOPs), training data exposure, memory, and deployment cost. The empirical chapters focus most directly on parameter-efficient adaptation, compute-controlled pretraining, and deployment-oriented adaptation budgets.

I developed this argument through two complementary empirical perspectives. The first concerns **external PEFT modules**: parameter-efficient fine-tuning (PEFT) components treated as reusable adaptation artefacts that can be described, ported, and organised under explicit budgets. The second concerns **internal circuits**: reusable internal computation, especially the internal structures that support in-context learning (ICL) as a form of reuse without parameter updates. Across both perspectives, the unifying concern is the same: how scarce resources should be organised around structures that carry useful behaviour.

The thesis uses modularity as an organising perspective for empirical questions

about reuse under constraints. On the external side, it describes PEFT components, tests paired-checkpoint transfer, and studies one cost-budgeted LoRA setting. On the internal side, it compares copy-related telemetry with targeted/random head ablations. The chapters do not establish that the same modularity conditions apply to both objects, nor do they provide a general method for identifying redundant computation.

8.2 Answers to the research questions

This thesis addressed four research questions.

Research Question 1: Can the parameters added by PEFT methods be treated as structurally and/or functionally modular? Chapter 4 addressed the structural part by introducing *PEFT-Ref*, a reference architecture and nine-property typology covering interface, insertion, workspace, integration, and related features. The typology is a descriptive basis for the seven surveyed methods, not a performance predictor or exhaustive taxonomy. Chapters 5 and 6 then test narrower functional and budget questions in their respective experimental settings.

Research Question 2: Can PEFT modules be reused within new host models, and what limits their portability? Chapter 5 operationalised *portability* between corresponding same-scale T5 v1.1 and Flan-T5 checkpoints. These hosts share architecture, tokenizer, tensor shapes, and training lineage; they are not independently trained same-architecture models. Ported modules have higher aggregate means than from-scratch and tensor-wise distribution-matched controls in the controlled sentiment study, while the full-grid outcomes vary with technique, direction, task/dataset setting, and budget. Architectural compatibility makes direct import possible but is not sufficient in general: independently trained hosts may have unaligned parameter spaces. The answer is therefore limited to graded, budget-sensitive transfer within the aligned checkpoint family studied here.

Research Question 3: How can adaptation resources be organised under explicit budgets? Chapter 6 tested Budgeted LoRA in a fixed warm-started 6-layer-student distillation setup. At $F = 0.4$, it has perplexity close to KD LoRA with the same teacher and lower adapted-module cost proxies; $F = 0.0$ reports larger proxy savings and worse perplexity. These sampled points are not claimed to form a Pareto frontier, and the 0.13B-teacher KD Full student is a stronger absolute-perplexity baseline than the larger-teacher budgeted variants. The controller orders removals by cost alone and has no quality-importance score, so it is an importance-agnostic baseline rather than a demonstrated optimal allocation. Higher ICL-composite means at larger teacher scales are specific to the copied, frozen-backbone setup and may reflect its altered optimisation parameterisation.

Research Question 4: How can internal structure supporting in-context learning be identified under matched compute? Chapter 7 used Bi-Induct to compare behavioural endpoints, copy-head telemetry, matched targeted/random ablation, and natural-text perplexity under fixed-shape nominal compute. Causal intervention is already a core requirement of mechanistic interpretability; the chapter’s narrower contribution is the joint application of these four checks in one matched-compute case study. Bi-Induct changes copy-related signatures without a consistent ICL gain, and the selected natural-only heads are generally more sensitive to targeted ablation under the tested protocol. The result is conditional on the random-token objective, unmasked first-span loss, top-one-per-layer localisation, three model scales, and function-probe endpoint.

8.3 Synthesis: Modularity for reuse under constraints

The thesis contributions are most useful when read together rather than as isolated chapter results. They support four practical conclusions about how large pretrained

models can be reused more effectively under constraints.

First, reusable PEFT modules need explicit structural descriptions. Before a PEFT component can be treated as reusable, its boundary, insertion point, workspace, integration form, and compatibility requirements must be made explicit. This is the role of PEFT-Ref in Chapter 4: it turns PEFT methods from parameter-saving implementation choices into comparable reusable artefacts.

Second, PEFT-module reuse must be tested rather than assumed. Within corresponding T5 v1.1/Flan-T5 checkpoints, Chapter 5 observes matched-budget benefits in some conditions and substantial variation by technique, direction, task/dataset setting, and budget. The result is conditional on the hosts’ shared training lineage and should not be extended to independently trained models without further evidence.

Third, Chapter 6 adds dense-retention budget and post-training consolidation as variables in one LoRA-distillation setup. The reported settings provide within-method quality-proxy-cost comparisons, while KD Full remains the hard quality baseline and the cost-only controller remains to be compared with importance-aware allocation.

Fourth, the Bi-Induct case study follows the established principle that visible signatures do not establish a causal circuit. Reporting behavioural capability, telemetry, targeted/random ablation, and natural-text quality together makes the negative result interpretable under the study’s matched-compute constraints.

PEFT classification, transfer, LoRA distillation, pruning, induction heads, and causal circuit tests all precede this work. The thesis contribution is a specific empirical sequence: a nine-property PEFT description, paired T5-family portability experiments, a cost-budgeted LoRA distillation study, and a four-axis Bi-Induct case study. External PEFT modules and inferred internal circuits are not treated as the same kind of object, and no direct causal bridge between them is established.

Part II develops three related but bounded analyses. PEFT-Ref supplies a structural description; PEFT-Port measures transfer within corresponding T5-family

checkpoints; and Budgeted LoRA samples quality–cost outcomes for a cost-only controller in one distillation setup. Together, they identify interfaces, host alignment, budget, and baseline choice as conditions that must be reported when making PEFT reuse claims.

Part III studies one candidate internal mechanism. In the Bi-Induct experiments, more visible copy-related signatures do not correspond to a consistent ICL benefit or greater sensitivity to the selected-head ablation. This is a case-specific application of the field’s established distinction between correlational telemetry and causal evidence.

The connection across the parts is methodological rather than mechanistic. External modules are designed components whose transfer and costs can be measured directly; internal circuits are inferred structures requiring intervention. Both analyses ask what evidence supports reuse under a stated constraint, but they use different objects, baselines, and validation criteria.

The relation between Chapters 6 and 7 remains a future hypothesis. Budgeted LoRA records setup-specific differences in function-probe means; Bi-Induct separately tests copy-head telemetry and ablation. The thesis does not show that a particular external adaptation preserves or damages a particular internal circuit.

The thesis also points to a broader design space involving the internalisation and externalisation of different capabilities. These approaches can be combined. Some capabilities may be best represented inside the model, either as dense capability encoded in weights or as load-bearing circuit structure that supports reuse without parameter updates. Other capabilities may be better externalised into detachable PEFT modules, retrieved context, memory systems, tools, control policies, or other harness-level structures. A central future question concerns where different reusable capabilities should be represented, adapted, stored, or invoked across internal circuits, external PEFT modules, data interventions, and broader system-level scaffolds under fixed constraints.

The resulting thesis position is deliberately limited: claims about reusable com-

ponents require explicit interfaces, scope, budgets, hard baselines, and functional tests appropriate to the object. For designed PEFT modules, those tests include paired-host transfer and deployment-cost accounting; for inferred circuits, they include causal intervention in addition to telemetry.

8.4 Limitations and future directions

The thesis has several limitations, each of which points to natural future work.

First, PEFT-Ref covers seven methods and nine selected properties. The selection is motivated but not exhaustive, and the typology does not replace mathematical method definitions or predict performance. Future work should test its coverage and inter-annotator reliability on newer PEFT families.

Second, PEFT-Port uses corresponding same-scale T5 v1.1/Flan-T5 checkpoints that share architecture, tokenizer, shapes, and training lineage. It does not test independently trained same-architecture hosts, where parameter bases may be unaligned. The 99% recovery rule is an operational threshold rather than statistical equivalence, and the full grid uses only two seeds. Future work should test independently trained hosts with explicit alignment methods, more model families, more seeds, and uncertainty-aware recovery criteria.

Third, Budgeted LoRA uses an importance-agnostic controller with removal score $s_m = c_m$. It is not compared with magnitude-, gradient-, activation-, Fisher-, or validation-loss-based importance, and it cannot show that the retained modules are behaviourally well chosen. KD Full, including the 0.13B-teacher case, remains the hard absolute-perplexity baseline. The reported training and inference values are adapted-module proxies rather than end-to-end wall-clock or serving measurements. Future work should add importance-aware controllers, unpruned and smaller-student deployment baselines, hardware measurements, and alternative student initialisations.

Fourth, Bi-Induct matches nominal FLOPs but not non-padding supervised tokens or natural-text exposure. Its random first span is included in the language-

model loss even though it is largely unpredictable; masking that loss would retain gradients from the predictable second span and provide a cleaner objective. The top-2% rule reduces to one head per layer, no larger top- k sensitivity is reported, and the causal test covers one head-output intervention and function-probe family. Future work should compare real-text spans, masked-first-span loss, token-matched controls, broader localisation rules, and interventions over residual and MLP components.

Fifth, no causal bridge is established between external PEFT choices and internal circuits. A direct study would apply circuit measurements and interventions to students produced by the alternative adaptation methods under matched initialisation and parameterisation controls.

A further future direction concerns agentic and continual-learning systems. In agentic systems, useful behaviour is often assembled across multiple levels: internal model computation, external PEFT modules, retrieved context, memory, tools, and control policies. In continual-learning settings, the central challenge is to update different capabilities incrementally without repeatedly rewriting the full model or destroying previously useful structure. The forms of modularity studied in this thesis suggest one route forward: capability should be selectively added, preserved, and invoked across system levels, rather than globally rewritten whenever the objective changes.

More broadly, the thesis suggests a future research direction in which reusable capability is studied across PEFT modules, internal circuits, data interventions, and harness-level scaffolds. These should not be treated as isolated techniques, but as different levers for shaping where different capabilities are represented and how they can be reused under fixed budgets.

8.5 Final remarks

This thesis examines how evidence for reuse changes when parameters, training exposure, or deployment computation are constrained. Its empirical results are

specific to the surveyed PEFT methods, paired T5-family checkpoints, one LoRA distillation setup, and one induction-focused data intervention.

Across those settings, three reporting requirements recur: define the component and interface precisely, compare it with hard baselines under an explicit budget, and distinguish structural or correlational evidence from functional and causal evidence. This is the practical role of modularity in the thesis, rather than a claim that all useful LLM capability is cleanly decomposable.

A future programme can test where capabilities are best adapted or invoked—in PEFT modules, internal computation, data interventions, or external systems—but the experiments here do not resolve that allocation. They provide scoped protocols and negative as well as positive results for evaluating such claims under constraints.

Bibliography

Aghajanyan, A., Zettlemoyer, L., and Gupta, S. (2020). Intrinsic dimensionality explains the effectiveness of language model fine-tuning.

Alon, U. (2006). *An Introduction to Systems Biology*. Chapman and Hall/CRC.

Amini, A., Gabriel, S., Lin, P., Koncel-Kedziorski, R., Choi, Y., and Hajishirzi, H. (2019). Mathqa: Towards interpretable math word problem solving with operation-based formalisms.

Arora, S., Cohen, N., Hu, W., and Luo, Y. (2019). Implicit regularization in deep matrix factorization.

Auda, G. and Kamel, M. (1999). Modular neural networks: A Survey. *International Journal of Neural Systems*, 09(02):129–151.

Azimi, R., Rishav, R., Teichmann, M., and Kahou, S. E. (2024). Kd-lora: A hybrid approach to efficient fine-tuning with lora and knowledge distillation.

Baldwin, C. Y. and Clark, K. B. (2000). *Design Rules, Volume 1: The Power of Modularity*. The MIT Press.

Bansal, Y., Ghorbani, B., Garg, A., Zhang, B., Krikun, M., Cherry, C., Neyshabur, B., and Firat, O. (2022). Data scaling laws in nmt: The effect of noise and architecture.

Belz, A. (2021). Quantifying reproducibility in nlp and ml.

-
- Belz, A., Popovic, M., and Mille, S. (2022). Quantified reproducibility assessment of NLP results. In Muresan, S., Nakov, P., and Villavicencio, A., editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16–28, Dublin, Ireland. Association for Computational Linguistics.
- Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09*, page 41–48, New York, NY, USA. Association for Computing Machinery.
- Bisk, Y., Zellers, R., Bras, R. L., Gao, J., and Choi, Y. (2020). Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*.
- Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N., Chen, A., Creel, K., Davis, J. Q., Demszky, D., Donahue, C., Doumbouya, M., Durmus, E., Ermon, S., Etchemendy, J., Ethayarajh, K., Fei-Fei, L., Finn, C., Gale, T., Gillespie, L., Goel, K., Goodman, N., Grossman, S., Guha, N., Hashimoto, T., Henderson, P., Hewitt, J., Ho, D. E., Hong, J., Hsu, K., Huang, J., Icard, T., Jain, S., Jurafsky, D., Kalluri, P., Karamcheti, S., Keeling, G., Khani, F., Khattab, O., Koh, P. W., Krass, M., Krishna, R., Kuditipudi, R., Kumar, A., Ladhak, F., Lee, M., Lee, T., Leskovec, J., Levent, I., Li, X. L., Li, X., Ma, T., Malik, A., Manning, C. D., Mirchandani, S., Mitchell, E., Munyikwa, Z., Nair, S., Narayan, A., Narayanan, D., Newman, B., Nie, A., Niebles, J. C., Nilforoshan, H., Nyarko, J., Ogut, G., Orr, L., Papadimitriou, I., Park, J. S., Piech, C., Portelance, E., Potts, C., Raghunathan, A., Reich, R., Ren, H., Rong, F., Roohani, Y., Ruiz, C., Ryan, J., Ré, C., Sadigh, D., Sagawa, S., Santhanam, K., Shih, A., Srinivasan, K., Tamkin, A., Taori, R., Thomas, A. W., Tramèr, F., Wang, R. E., Wang, W., Wu, B., Wu, J., Wu, Y.,

-
- Xie, S. M., Yasunaga, M., You, J., Zaharia, M., Zhang, M., Zhang, T., Zhang, X., Zhang, Y., Zheng, L., Zhou, K., and Liang, P. (2022). On the opportunities and risks of foundation models.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Nee-lakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.
- Busbridge, D., Shidani, A., Weers, F., Ramapuram, J., Littwin, E., and Webb, R. (2025). Distillation scaling laws.
- Carroll, S. B. (2001). Chance and necessity: the evolution of morphological complexity and diversity. *Nature*, 409(6823):1102–1109.
- Chen, G., Liu, F., Meng, Z., and Liang, S. (2022). Revisiting parameter-efficient tuning: Are we really there yet? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2612–2626, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.
- Chen, S., Sheen, H., Wang, T., and Yang, Z. (2024). Unveiling induction heads: Provable training dynamics and feature learning in transformers.
- Cheng, H., Zhang, M., and Shi, J. Q. (2024). A survey on deep neural network pruning-taxonomy, comparison, analysis, and recommendations.
- Chou, H.-H., Gieshoff, C., Maly, J., and Rauhut, H. (2023). Gradient descent for deep matrix factorization: Dynamics and implicit bias towards low rank.
- Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen,

-
- X., Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., Valter, D., Narang, S., Mishra, G., Yu, A., Zhao, V., Huang, Y., Dai, A., Yu, H., Petrov, S., Chi, E. H., Dean, J., Devlin, J., Roberts, A., Zhou, D., Le, Q. V., and Wei, J. (2022). Scaling instruction-finetuned language models.
- Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. (2019). BoolQ: Exploring the surprising difficulty of natural yes/no questions. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.
- Clark, P., Cowhey, I., Etzioni, O., Khot, T., Sabharwal, A., Schoenick, C., and Tafjord, O. (2018). Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457.
- Clune, J., Mouret, J.-B., and Lipson, H. (2013). The evolutionary origins of modularity. *Proceedings of the Royal Society B: Biological Sciences*, 280(1755):20122863.
- Cobbe, K., Kosaraju, V., Bavarian, M., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. (2021). Training verifiers to solve math word problems.
- Conmy, A., Mavor-Parker, A. N., Lynch, A., Heimersheim, S., and Garriga-Alonso, A. (2023). Towards automated circuit discovery for mechanistic interpretability.
- Dai, D., Dong, L., Hao, Y., Sui, Z., Chang, B., and Wei, F. (2022). Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8493–8502, Dublin, Ireland. Association for Computational Linguistics.
- Davidovich, O. and Ringel, Z. (2026). Algorithmic capture, computational complexity, and inductive bias of infinite transformers.

-
- Ding, N., Qin, Y., Yang, G., Wei, F., Yang, Z., Su, Y., Hu, S., Chen, Y., Chan, C.-M., Chen, W., Yi, J., Zhao, W., Wang, X., Liu, Z., Zheng, H.-T., Chen, J., Liu, Y., Tang, J., Li, J., and Sun, M. (2023). Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235.
- Dolan, W. B. and Brockett, C. (2005). Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.
- Edelman, E., Tsilivis, N., Edelman, B. L., eran malach, and Goel, S. (2024). The evolution of statistical induction heads: In-context learning markov chains. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Fujii, K., Tajima, Y., Mizuki, S., Shimada, H., Shiotani, T., Saito, K., Ohi, M., Kawamura, M., Nakamura, T., Okamoto, T., Ishida, S., Hattori, K., Ma, Y., Takamura, H., Yokota, R., and Okazaki, N. (2025). Rewriting pre-training data boosts llm performance in math and code.
- Gao, L., Biderman, S., Black, S., Golding, L., Hoppe, T., Foster, C., Phang, J., He, H., Thite, A., Nabeshima, N., Presser, S., and Leahy, C. (2020). The pile: An 800gb dataset of diverse text for language modeling.
- Geva, M., Schuster, R., Berant, J., and Levy, O. (2021). Transformer feed-forward layers are key-value memories.
- Gliwa, B., Mochol, I., Biesek, M., and Wawer, A. (2019). SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 70–79, Hong Kong, China. Association for Computational Linguistics.
- Gloeckle, F., Idrissi, B. Y., Rozière, B., Lopez-Paz, D., and Synnaeve, G. (2024). Better & faster large language models via multi-token prediction.

-
- Goldowsky-Dill, N., MacLeod, C., Sato, L., and Arora, A. (2023). Localizing model behavior with path patching.
- Gu, Y., Dong, L., Wei, F., and Huang, M. (2023). Pre-training to learn in context.
- Gupta, R., Arcuschin, I., Kwa, T., and Garriga-Alonso, A. (2024). Interp-bench: Semi-synthetic transformers for evaluating mechanistic interpretability techniques. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.
- Han, Z., Gao, C., Liu, J., Zhang, J., and Zhang, S. Q. (2024). Parameter-efficient fine-tuning for large models: A comprehensive survey.
- He, J., Zhou, C., Ma, X., Berg-Kirkpatrick, T., and Neubig, G. (2022). Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*.
- Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. (2021). Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network.
- Hintze, A. and Adami, C. (2008). Evolution of complex modular biological networks. *PLoS Computational Biology*, 4(2):e23.
- Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O., and Sifre, L. (2022). Training compute-optimal large language models.
- Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer

-
- learning for NLP. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, page 2799. PMLR.
- Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., and Chen, W. (2021). Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685.
- Hwang, I., Park, H., Lee, Y., Yang, J., and Maeng, S. (2024). Pc-lora: Low-rank adaptation for progressive model compression with knowledge distillation.
- Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. (2023). Mistral 7b.
- Jiang, J., Shu, Y., Wang, J., and Long, M. (2022). Transferability in deep learning: A survey.
- Jiao, X., Yin, Y., Shang, L., Jiang, X., Chen, X., Li, L., Wang, F., and Liu, Q. (2020). Tinybert: Distilling bert for natural language understanding.
- Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. (2017). Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada. Association for Computational Linguistics.
- Kadelka, C., Wheeler, M., Veliz-Cuba, A., Murrugarra, D., and Laubenbacher, R. (2023). Modularity of biological systems: a link between structure and function. *bioRxiv*.
- Kahardipraja, P., Achibat, R., Wiegand, T., Samek, W., and Lapuschkin, S. (2026). The atlas of in-context learning: How attention heads shape in-context retrieval augmentation. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

-
- Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models.
- Karimi Mahabadi, R., Henderson, J., and Ruder, S. (2021). Compacter: Efficient low-rank hypercomplex adapter layers. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 1022–1035. Curran Associates, Inc.
- Kashtan, N. and Alon, U. (2005). Spontaneous evolution of modularity and network motifs. *Proceedings of the National Academy of Sciences*, 102(39):13773–13778.
- Kashtan, N., Noor, E., and Alon, U. (2007). Varying environments can speed up evolution. *Proceedings of the National Academy of Sciences*, 104(34):13711–13716.
- Kim, J., Kwon, S., Choi, J. Y., Park, J., Cho, J., Lee, J. D., and Ryu, E. K. (2025). Task diversity shortens the icl plateau.
- Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020). Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.
- Lee, J., Tang, R., and Lin, J. (2019). What would elsa do? freezing layers during transformer fine-tuning.
- Lester, B., Al-Rfou, R., and Constant, N. (2021). The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Li, X. L. and Liang, P. (2021). Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

-
- Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., and Raffel, C. (2022). Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning.
- Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. (2017). Learning efficient convolutional networks through network slimming.
- Longpre, S., Hou, L., Vu, T., Webson, A., Chung, H. W., Tay, Y., Zhou, D., Le, Q. V., Zoph, B., Wei, J., and Roberts, A. (2023). The flan collection: Designing data and methods for effective instruction tuning.
- Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *International Conference on Learning Representations*.
- Louizos, C., Welling, M., and Kingma, D. P. (2018). Learning sparse neural networks through l_0 regularization.
- MacCormack, A., Rusnak, J., and Baldwin, C. Y. (2007). The impact of component modularity on design evolution: Evidence from the software industry. *SSRN Electron. J.*
- Maini, P., Seto, S., Bai, H., Grangier, D., Zhang, Y., and Jaitly, N. (2024). Rephrasing the web: A recipe for compute and data-efficient language modeling.
- Mansourian, A. M., Ahmadi, R., Ghafouri, M., Babaei, A. M., Golezani, E. B., Ghamchi, Z. Y., Ramezani, V., Taherian, A., Dinashi, K., Miri, A., and Kasaei, S. (2025). A comprehensive survey on knowledge distillation.
- Mao, Y., Mathias, L., Hou, R., Almahairi, A., Ma, H., Han, J., Yih, S., and Khabsa, M. (2022). UniPELT: A unified framework for parameter-efficient language model tuning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6253–6264, Dublin, Ireland. Association for Computational Linguistics.

-
- Marelli, M., Menini, S., Baroni, M., Bentivogli, L., Bernardi, R., and Zamparelli, R. (2014). A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC’14)*, pages 216–223, Reykjavik, Iceland. European Language Resources Association (ELRA).
- Meng, K., Bau, D., Andonian, A., and Belinkov, Y. (2023). Locating and editing factual associations in gpt.
- Meunier, D., Lambiotte, R., and Bullmore, E. (2010). Modular and hierarchically modular organization of brain networks. *Frontiers in Neuroscience*, 4.
- Michel, P., Levy, O., and Neubig, G. (2019). Are sixteen heads really better than one?
- Mihaylov, T., Clark, P., Khot, T., and Sabharwal, A. (2018). Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.
- Mille, S., Sabry, M., and Belz, A. (2024). DCU-NLG-small at the GEM’24 data-to-text task: Rule-based generation and post-processing with t5-base. In Mille, S. and Clinciu, M.-A., editors, *Proceedings of the 17th International Natural Language Generation Conference: Generation Challenges*, pages 84–91, Tokyo, Japan. Association for Computational Linguistics.
- Min, S., Lyu, X., Holtzman, A., Artetxe, M., Lewis, M., Hajishirzi, H., and Zettlemoyer, L. (2022). Rethinking the role of demonstrations: What makes in-context learning work?
- Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., and Gebru, T. (2018). Model cards for model reporting. *CoRR*, abs/1810.03993.
- Muralidharan, S., Sreenivas, S. T., Joshi, R., Chochowski, M., Patwary, M., Shoeybi, M., Catanzaro, B., Kautz, J., and Molchanov, P. (2024). Compact language models via pruning and knowledge distillation.

-
- Nguyen, T., Li, Y., Golovneva, O., Zettlemoyer, L., Oh, S., Schmidt, L., and Li, X. (2025). Recycling the web: A method to enhance pre-training data quality and quantity for language models.
- Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., and Carter, S. (2020a). Zoom in: An introduction to circuits. *Distill*. <https://distill.pub/2020/circuits/zoom-in>.
- Olah, C., Cammarata, N., Voss, C., Schubert, L., and Goh, G. (2020b). Naturally occurring equivariance in neural networks. *Distill*. <https://distill.pub/2020/circuits/equivariance>.
- Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., Conerly, T., Drain, D., Ganguli, D., Hatfield-Dodds, Z., Hernandez, D., Johnston, S., Jones, A., Kernion, J., Lovitt, L., Ndousse, K., Amodei, D., Brown, T., Clark, J., Kaplan, J., McCandlish, S., and Olah, C. (2022). In-context learning and induction heads. Transformer Circuits Thread, <https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/>. Also available as arXiv:2209.11895.
- Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.
- Pang, B. and Lee, L. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL’05)*, pages 115–124, Ann Arbor, Michigan. Association for Computational Linguistics.
- Paperno, D., Kruszewski, G., Lazaridou, A., Pham, Q. N., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. (2016). The lambda dataset.
- Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058.

-
- Pfeiffer, J., Ruder, S., Vulić, I., and Ponti, E. M. (2023). Modular deep learning.
- Pfeiffer, J., Vulić, I., Gurevych, I., and Ruder, S. (2020). MAD-X: An Adapter-Based Framework for Multi-Task Cross-Lingual Transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Online. Association for Computational Linguistics.
- Qi, W., Ruan, Y.-P., Zuo, Y., and Li, T. (2022). Parameter-efficient tuning on layer normalization for pre-trained language models.
- Qiu, S., Potapczynski, A., Finzi, M., Goldblum, M., and Wilson, A. G. (2024). Compute better spent: Replacing dense layers with structured matrices.
- Rai, D., Zhou, Y., Feng, S., Saparov, A., and Yao, Z. (2025). A practical review of mechanistic interpretability for transformer-based language models.
- Rein, D., Hou, B. L., Stickland, A. C., Petty, J., Pang, R. Y., Dirani, J., Michael, J., and Bowman, S. R. (2023). Gpqa: A graduate-level google-proof q&a benchmark.
- Rücklé, A., Geigle, G., Glockner, M., Beck, T., Pfeiffer, J., Reimers, N., and Gurevych, I. (2021). AdapterDrop: On the efficiency of adapters in transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7930–7946, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.
- Ruder, S. (2017). An overview of multi-task learning in deep neural networks.
- Sabry, M. and Belz, A. (2023). Peft-ref: A modular reference architecture and typology for parameter-efficient finetuning techniques.
- Sabry, M. and Belz, A. (2024). Assessing the portability of parameter matrices trained by parameter-efficient finetuning methods. In Graham, Y. and Purver, M., editors, *Findings of the Association for Computational Linguistics: EACL 2024*, pages 1548–1556, St. Julian’s, Malta. Association for Computational Linguistics.

-
- Sabry, M. and Belz, A. (2026a). Budgeted lora: Distillation as structured compute allocation for efficient inference.
- Sabry, M. and Belz, A. (2026b). Induction signatures are not enough: A matched-compute study of load-bearing structure in in-context learning.
- Şahin, K., Feucht, S., Belfki, A., Brinkmann, J., Mueller, A., Bau, D., and Wendler, C. (2026). In-context learning without copying.
- Sakaguchi, K., Bras, R. L., Bhagavatula, C., and Choi, Y. (2019). Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*.
- Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2020a). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter.
- Sanh, V., Wolf, T., and Rush, A. M. (2020b). Movement pruning: Adaptive sparsity by fine-tuning.
- Sharma, L., Graesser, L., Nangia, N., and Evcı, U. (2019). Natural language understanding with the quora question pairs dataset. *CoRR*, abs/1907.01041.
- Shinnick, Z., Jiang, L., Saratchandran, H., van den Hengel, A., and Teney, D. (2025). Transformers pretrained on procedural data contain modular structures for algorithmic reasoning.
- Singh, A. K., Moskovitz, T., Hill, F., Chan, S. C., and Saxe, A. M. (2024). What needs to go right for an induction head? A mechanistic study of in-context learning circuits and their formation. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F., editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 45637–45662. PMLR.
- Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural*

Language Processing, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Soviany, P., Ionescu, R. T., Rota, P., and Sebe, N. (2022). Curriculum learning: A survey.

Su, Y., Wang, X., Qin, Y., Chan, C.-M., Lin, Y., Wang, H., Wen, K., Liu, Z., Li, P., Li, J., Hou, L., Sun, M., and Zhou, J. (2022). On transferability of prompt tuning for natural language processing. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3949–3969, Seattle, United States. Association for Computational Linguistics.

Subramanyam, A., Chen, Y., and Grossman, R. L. (2026). Scaling laws revisited: Modeling the role of data quality in language model pretraining.

Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q. V., Chi, E. H., Zhou, D., and Wei, J. (2022). Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Talmor, A., Herzig, J., Lourie, N., and Berant, J. (2019). CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

Tempero, E., Blincoe, K., and Lottridge, D. (2023). An experiment on the effects of modularity on code modification and understanding. In *Proceedings of the 25th Australasian Computing Education Conference, ACE '23*, page 105–112, New York, NY, USA. Association for Computing Machinery.

-
- Todd, E., Li, M. L., Sharma, A. S., Mueller, A., Wallace, B. C., and Bau, D. (2024). Function vectors in large language models.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- Veitsman, Y., Jobanputra, M., Sarrof, Y., Bakalova, A., Demberg, V., Pavlick, E., and Hahn, M. (2025). Born a transformer – always a transformer?
- Vig, J. (2019). A multiscale visualization of attention in the transformer model.
- Vu, T., Lester, B., Constant, N., Al-Rfou’, R., and Cer, D. (2022). SPoT: Better frozen model adaptation through soft prompt transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5039–5059, Dublin, Ireland. Association for Computational Linguistics.
- Wagner, G. P., Pavlicev, M., and Cheverud, J. M. (2007). The road to modularity. *Nature Reviews Genetics*, 8(12):921–931.
- Wang, K. R., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. (2023). Interpretability in the wild: a circuit for indirect object identification in GPT-2 small. In *The Eleventh International Conference on Learning Representations*.
- Wang, R., Ghosh, S., Cox, D. D., Antognini, D., Oliva, A., Feris, R., and Karlinsky, L. (2024). Trans-LoRA : towards data-free transferable parameter efficient finetuning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. (2020a). Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers.

-
- Wang, Z., Wohlwend, J., and Lei, T. (2020b). Structured pruning of large language models. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, page 6151–6162. Association for Computational Linguistics.
- Wei, J., Bosma, M., Zhao, V., Guu, K., Yu, A. W., Lester, B., Du, N., Dai, A. M., and Le, Q. V. (2022). Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.
- Wei, J., Wei, J., Tay, Y., Tran, D., Webson, A., Lu, Y., Chen, X., Liu, H., Huang, D., Zhou, D., and Ma, T. (2023). Larger language models do in-context learning differently.
- Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. (2016). Learning structured sparsity in deep neural networks.
- Williams, A., Nangia, N., and Bowman, S. (2018). A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.
- Xu, P., Patwary, M., Prabhunoye, S., Adams, V., Prenger, R., Ping, W., Lee, N., Shoeybi, M., and Catanzaro, B. (2022). Evaluating parameter efficient learning for generation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 4824–4833, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.
- Yang, H., Cho, H., Zhong, Y., and Inoue, N. (2026). Unifying attention heads and task vectors via hidden state geometry in in-context learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Yang, R., Wu, T., Wang, J., Hu, P., Wu, Y.-C., Wong, N., and Yang, Y. (2025). Llm-neo: Parameter efficient knowledge distillation for large language models.

-
- Yeh, R. A., Hu, Y.-T., Hasegawa-Johnson, M., and Schwing, A. (2022). Equivariance discovery by learned parameter-sharing. In Camps-Valls, G., Ruiz, F. J. R., and Valera, I., editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 1527–1545. PMLR.
- Yin, K. and Steinhardt, J. (2025). Which attention heads matter for in-context learning?
- Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS’14, page 3320–3328, Cambridge, MA, USA. MIT Press.
- Zaken, E. B., Ravfogel, S., and Goldberg, Y. (2022). Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models.
- Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., and Choi, Y. (2019). Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.
- Zhang, D., Feng, T., Xue, L., Wang, Y., Dong, Y., and Tang, J. (2025). Parameter-efficient fine-tuning for foundation models.
- Zhang, F. and Nanda, N. (2024). Towards best practices of activation patching in language models: Metrics and methods.
- Zhang, Q., Chen, M., Bukharin, A., Karampatziakis, N., He, P., Cheng, Y., Chen, W., and Zhao, T. (2023). Adalora: Adaptive budget allocation for parameter-efficient fine-tuning.
- Zhao, H., Tan, H., and Mei, H. (2022). Tiny-attention adapter: Contexts are more important than the number of parameters. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6626–6638, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Zhao, T. Z., Wallace, E., Feng, S., Klein, D., and Singh, S. (2021). Calibrate before use: Improving few-shot performance of language models.

Appendix A

Research artefacts: Model cards

In addition to the written contributions of the thesis, I released a family of open-source language models intended to support reproducible experimentation on modularity, distillation, in-context learning, and compute-aware training. These releases function as research artefacts aligned with the central thesis theme of *reuse under constraints*: they provide controlled model checkpoints that other researchers can use to study how capability, mechanism, and efficiency interact under matched architectural and training conditions.

A.1 Overview and motivation

The released family, which I refer to as BI-INDUCT, consists of 12 decoder-only language models across three scales: 125M, 500M, and 1B parameters. For each scale, I release four training variants:

- **Baseline**: natural-text pretraining only;
- **Induction**: natural text plus replacement-based induction-style synthetic sequences;
- **Anti-Induction**: natural text plus replacement-based anti-induction-style synthetic sequences;

-
- **Balanced:** natural text plus a balanced mixture of induction-style and anti-induction-style replacement sequences.

The motivation for releasing these checkpoints is twofold. First, they provide reusable experimental infrastructure for replicating and extending the matched-compute studies in Chapter 7. Second, they support follow-up work on related questions developed elsewhere in the thesis, including distillation, PEFT-based interventions, and compute-constrained adaptation (Chapter 6).

A.2 Model family summary

All models in the Bi-Induct family are decoder-only Transformers trained under a controlled experimental framework. The family is designed to make cross-model comparisons meaningful: scales vary systematically, and within each scale the primary difference is the pretraining data regime rather than a wholesale change in architecture or evaluation setup. This makes the releases useful for studying how targeted data interventions affect both behavioural capability and internal mechanistic structure.

A.3 Intended use

These models are intended for research use. In particular, they are suitable for:

- studying in-context learning under controlled pretraining conditions;
- probing mechanistic signatures such as induction-like behaviour;
- evaluating whether such signatures are load-bearing under targeted ablation;
- running distillation experiments from larger to smaller controlled hosts;
- testing PEFT and LoRA-based interventions under matched architectures;
- analysing the interaction between data regime, internal circuits, and compute-aware adaptation.

Scale	Variant	Repository
125M	Baseline	MohammedSabry/biinduct-125m-baseline
	Induction	MohammedSabry/biinduct-125m-induction
	Anti-Induction	MohammedSabry/biinduct-125m-anti-induction
	Balanced	MohammedSabry/biinduct-125m-balanced
500M	Baseline	MohammedSabry/biinduct-500m-baseline
	Induction	MohammedSabry/biinduct-500m-induction
	Anti-Induction	MohammedSabry/biinduct-500m-anti-induction
	Balanced	MohammedSabry/biinduct-500m-balanced
1B	Baseline	MohammedSabry/biinduct-1b-baseline
	Induction	MohammedSabry/biinduct-1b-induction
	Anti-Induction	MohammedSabry/biinduct-1b-anti-induction
	Balanced	MohammedSabry/biinduct-1b-balanced

Table A.1: Released checkpoints in the Bi-Induct model family on Hugging Face. The family spans three scales (125M, 500M, 1B) and four training regimes per scale (Baseline, Induction, Anti-Induction, Balanced), providing controlled research artefacts for reproducible experiments on in-context learning, mechanistic analysis, distillation, and modular adaptation.

A.4 Limitations and cautions

These releases are research models rather than general-purpose assistants. They are not instruction-tuned or safety-tuned for deployment-facing use, and they should be interpreted primarily as controlled experimental checkpoints. Their outputs may reflect limitations, biases, and failure modes inherited from the training data and from their relatively small scale compared to frontier systems. They are therefore best used for analysis, benchmarking, and follow-up experimentation, rather than as end-user-facing models.

A.5 Repositories

Table A.1 lists the released checkpoints in the Bi-Induct family and their corresponding Hugging Face repositories.

Appendix B

Supplementary Material for Chapter 4 (PEFT-Ref)

B.1 PEFT-Ref supplementary tables

B.1.1 Task performance evidence from prior work

Table B.1: Average PEFT technique accuracies on different tasks across datasets, as reported in prior comparative studies. Tiny-Attention Adapters are omitted due to the absence of comparative studies in the published literature. To preserve the required minimum font size, study context, training details, and reported accuracies are presented in separate parts.

Work	Task	Datasets
(Ding et al., 2023)	Sentiment Analysis	GLUE-SST2; ROTTEN_TOMATOES; FINANCIAL_PHRASEBANK; POEM_SENTIMENT; YELP_POLARITY
(Ding et al., 2023)	Classification/emotion	EMO; EMOTION; TWEET_EVAL-HATE; TWEET_EVAL-IRONY; TWEET_EVAL-OFFENSIVE; TWEET_EVAL-SENTIMENT; TWEET_EVAL-STANCE_ABORTION; TWEET_EVAL-STANCE_ATHEISM; TWEET_EVAL-STANCE_CLIMATE; TWEET_EVAL-STANCE_FEMINIST; TWEET_EVAL-STANCE_HILLARY
(Ding et al., 2023)	Natural Language Inference	ANLI; GLUE-MNLI; GLUE-QNLI; GLUE-RTE; SCITAIL; SUPERGLUE-RTE; SICK; SUPERGLUE-CB
(Ding et al., 2023)	Multiple-Choice QA	COSMOS_QA; DREAM; HELLASWAG; OPENBOOKQA; QASC; QUAREL; QUARTZ-NO_KNOWLEDGE; QUARTZ-WITH_KNOWLEDGE; RACE-HIGH; RACE-MIDDLE; SUPERGLUE-COPA; WINO_GRANDE; COMMONSENSE_QA; SCIQ; WIQA

Continued on next page

Table B.1 continued: study context

Work	Task	Datasets
(Ding et al., 2023)	Summarisation	SAMSUM; XSUM
(Liu et al., 2022)	Commonsense	H-Swag; COPA; StoryCloze; Winogrande
(Liu et al., 2022)	Word Sense Disambiguation	WiC
(Ding et al., 2023)	Hate-speech detection	THOS-DISABILITY; ETHOS-GENDER; ETHOS-NATIONAL_ORIGIN; ETHOS-RACE; ETHOS-RELIGION; ETHOS-DIRECTED_VS_GENERALIZED; HATE_SPEECH_OFFENSIVE; HATE_SPEECH; HATEXPLAIN

Table B.1 (continued): training details.

Work	Task	Training Details
(Ding et al., 2023)	Sentiment Analysis	Model: T5 base; Training Steps: 20k, except Prompt Tuning trained with 100K steps; best over batch size {16,32} and LR {1e-3,1e-4,5e-4}.
(Ding et al., 2023)	Classification/emotion	Model: T5 base; Training Steps: 20k, except Prompt Tuning trained with 100K steps; best over batch size {16,32} and LR {1e-3,1e-4,5e-4}.

Continued on next page

Table B.1 continued: training details

Work	Task	Training Details
(Ding et al., 2023)	Natural Language Inference	Model: T5 base; Training Steps: 20k, except Prompt Tuning trained with 100K steps; best over batch size {16,32} and LR {1e-3,1e-4,5e-4}.
(Ding et al., 2023)	Multiple-Choice QA	Model: T5 base; Training Steps: 20k, except Prompt Tuning trained with 100K steps; best over batch size {16,32} and LR {1e-3,1e-4,5e-4}.
(Ding et al., 2023)	Summarisation	Model: T5 base; Training Steps: 20k, except Prompt Tuning trained with 100K steps; best over batch size {16,32} and LR {1e-3,1e-4,5e-4}.
(Liu et al., 2022)	Commonsense	Model: T0-3b; Training Steps: 1k, except Compacter/Adapter trained with 500 steps; batch size {8}, LR {3e-3}, except Prompt Tuning {1e-3}.
(Liu et al., 2022)	Word Sense Disambiguation	Model: T0-3b; Training Steps: 1k, except Compacter/Adapter trained with 500 steps; batch size {8}, LR {3e-3}, except Prompt Tuning {1e-3}.
(Ding et al., 2023)	Hate-speech detection	Model: T5 base; Training Steps: 20k, except Prompt Tuning trained with 100K steps; best over batch size {16,32} and LR {1e-3,1e-4,5e-4}.

Table B.1 (continued): reported PEFT accuracies.

Work	Task	LoRA Prefix Prompt (IA) ³ Adapter Compacter					
(Ding et al., 2023)	Sentiment Analysis	93.09	92.83	85.48	–	92.06	–
(Ding et al., 2023)	Classification/emotion	68.70	67.21	52.95	–	68.31	–
(Ding et al., 2023)	Natural Language Inference	82.73	80.07	51.93	–	83.06	–
(Ding et al., 2023)	Multiple-Choice QA	58.67	53.93	46.93	–	56.11	–
(Ding et al., 2023)	Summarisation	35.44	33.61	30.35	–	35.38	–
(Liu et al., 2022)	Commonsense	66.23	58.95	61.05	68.01	63.75	65.65
(Liu et al., 2022)	Word Sense Disambiguation	54.86	52.51	52.51	54.23	54.70	55.33
(Ding et al., 2023)	Hate-speech detection	85.22	84.37	67.69	–	86.02	–

Appendix C

Supplementary Material for Chapter 5 (PEFT-Port)

C.1 Appendix roadmap

This appendix provides the supplementary experimental and evaluation details for Chapter 5. The main chapter defines the PEFT-Port protocol, operational portability criteria, and central results. The material below documents the experimental grids, host models and datasets, distribution-matched control, evaluation implementation, PEFT hyperparameters, training budgets, and computational resources needed to reproduce and interpret those results.

- Section C.2 documents the experimental and evaluation implementation.
- Section C.2.1 distinguishes the controlled-baseline and full-study grids.
- Section C.2.3 specifies the distribution-matched initialisation control.
- Section C.2.4 records prompt, decoding, output-normalisation, metric, and aggregation details.
- Sections C.2.5–C.2.7 provide PEFT, training, and computational-resource details.

Study	Scope	Pre-porting budgets		Post-porting evaluation points	Seeds
Controlled-baseline study	Sentiment analysis using SST-2 and Rotten Tomatoes; Base and Large host scales; ported, from-scratch, and distribution-matched conditions	{5K, 10K}	{5K, 10K}	{0, 0.5K, 1.5K, 3K} steps	{124, 5090, 675323}
Full portability study	Four task families across seven datasets; Base, Large, and XL host scales; ported and from-scratch conditions	{5K, 20K, 40K}	{10K, 20K, 40K}	{0, 0.5K, 1.5K, 3K, 5K} steps for imported modules	{124, 5090}

Table C.1: Summary of the two PEFT-Port experimental grids. Zero denotes immediate evaluation after import, without post-porting updates.

C.2 Experimental and evaluation details

The PEFT-Port protocol and operational criteria are defined in Sections 5.2.1 and 5.2.2. This section provides the supplementary implementation details that are intentionally kept outside the main chapter.

C.2.1 Experimental grids and host-model pairs

The chapter reports two complementary experimental grids. The *controlled-baseline study* isolates whether the observed porting advantage can be explained by favourable parameter statistics at initialisation. The *full portability study* expands the analysis across tasks, datasets, model scales, porting directions, and pre-/post-porting budgets.

Porting is restricted to corresponding same-scale raw and instruction-tuned hosts: T5 v1.1 Base $\leftrightarrow$ Flan-T5 Base, T5 v1.1 Large $\leftrightarrow$ Flan-T5 Large, and, in the full study, T5 v1.1 XL $\leftrightarrow$ Flan-T5 XL. No cross-scale module porting is evaluated. Each same-scale pair is evaluated in both raw $\rightarrow$ instruction-tuned and instruction-tuned $\rightarrow$ raw directions.

C.2.2 Host models, datasets, and split policy

All host models and datasets are loaded through the Hugging Face model and dataset interfaces used in the original 2023 implementation.

Host models.

- **T5 v1.1 (raw):** Base, Large, and XL variants.
- **Flan-T5 (instruction-tuned):** Base, Large, and XL variants.

Datasets and splits. For the full portability study, I use the standard dataset splits provided through the Hugging Face Datasets Hub. Where labelled test sets are available, the provided training, development, and test splits are used for their corresponding purposes. Where test-set labels are unavailable, as for SST-2, MNLI, and QQP, I use the original development split for final evaluation and reserve 10% of the original training split as a new development set. The remaining 90% is used for training. This preserves a separate development set for model evaluation and checkpoint selection without using the final evaluation split during training.

- **Sentiment analysis:** SST-2 and Rotten Tomatoes.
- **Natural language inference:** MNLI and SICK.
- **Paraphrase detection:** QQP and MRPC.
- **Summarisation:** SAMSum.

C.2.3 Distribution-matched initialisation control

The controlled-baseline study compares ported modules with both standard from-scratch modules and a distribution-matched initialisation control. The purpose of the distribution-matched condition is to test whether a ported module’s advantage can be explained by favourable first- and second-order parameter statistics alone, rather than by transferred task-specific information.

For each run, I first instantiate a fresh importing-side module using the same PEFT technique as the ported module. I then iterate over corresponding source and target parameter tensors. For each source tensor, I compute its empirical mean and variance and independently sample every element of the corresponding target tensor from a Gaussian distribution with that mean and standard deviation. Thus, moment matching is performed separately for each corresponding parameter tensor, rather than once over the complete module or layer. The newly sampled module is then trained using the same importing-side optimisation settings and post-porting budget as the ported and default-initialisation controls. A fresh, reproducible sample is generated for each experimental configuration and random seed.

Formally, for a source parameter tensor W_s and corresponding fresh target tensor W_t , the control replaces the target values with

$$W_t[i] \sim \mathcal{N}(\mu(W_s), \sigma^2(W_s))$$

independently over tensor elements i .

C.2.4 Prompting, decoding, output normalisation, and metrics

All tasks are formulated as sequence-to-sequence prediction without natural-language task descriptions or instruction prefixes. For the T5-family hosts used in the study, the separator inserted into classification inputs is the T5 end-of-sequence token, rendered below as `</s>`.

Generation. Final evaluation uses the Hugging Face `generate` interface with batch size one, the task-specific maximum generation lengths in Table C.2, and `early_stopping=True`. The code does not explicitly set sampling, beam-search, or other decoding arguments such as `do_sample` or `num_beams`; these therefore inherit the installed Transformers version and checkpoint generation defaults. Generation stops at the task-specific maximum length or the model’s end-of-sequence condition.

Task family	Input format	Target outputs	Max. input	Max. generation
Sentiment analysis	text </s> this is	terrible great	128	2
Natural language inference	sentence_1 </s> sentence_2 </s> :	entailment neutral contradiction	70 (SICK) 512 (MNLI)	8
Paraphrase detection	sentence_1 </s> sentence_2 </s> :	QQP: not duplicate duplicate MRPC: not equivalent equivalent	350 (QQP) 80 (MRPC)	4
SAMSum summarisation	Raw dialogue text	Free-text reference summary	512	128

Table C.2: Prompt formats, target outputs, and maximum sequence lengths used in the PEFT-Port evaluation code. Maximum lengths are measured in tokenizer tokens.

Classification output processing. Predictions and references are decoded with special tokens removed and Hugging Face tokenisation-space clean-up enabled. Classification outputs are then lower-cased and compared by exact string equality with the allowed labels shown in Table C.2. No additional punctuation removal or explicit whitespace-stripping rule is applied beyond the tokenizer’s decoding clean-up. Unrecognised non-empty generations are mapped to an invalid class. When metric computation fails because invalid outputs are mixed with valid classes, the evaluation wrapper records a score of zero for that configuration. Empty predictions are removed before scoring when they account for at most 10% of the decoded references; otherwise they remain invalid and can cause the configuration to receive a zero score. Accuracy is computed using the Hugging Face `datasets` accuracy metric.

Summarisation. SAMSum predictions and references are decoded with special tokens removed and tokenisation-space clean-up enabled. The implementation loads the Hugging Face `datasets` ROUGE metric, computes ROUGE with stemming enabled, and selects the third aggregate element returned for ROUGE-L. The stored value is a three-component precision/recall/F-measure result, and the downstream analysis uses its F-measure component.

Aggregation and uncertainty. The controlled-baseline aggregate table is computed after excluding the XL hosts. For each dataset scenario, PEFT technique, and importing-side condition, the reported mean and sample variance are calculated over 72 pooled run-level scores: four Base/Large target hosts (raw and instruction-tuned), two pre-porting budgets, three post-porting budgets, and three random seeds. Consequently, the variance reflects heterogeneity across hosts, budgets, and seeds; it is not solely seed-level uncertainty. Runs whose evaluation failed are stored with score zero and are included in these pooled aggregates.

The full portability study uses two random seeds, 124 and 5090. Reported full-grid performance results and heatmap ratios are averaged across these two seeds. The portability percentages are reported over configuration-level cases after aggregating the corresponding repeated-seed runs, following the case definition in Section 5.2.2. Zero-shot non-collapse counts configurations whose aggregated zero-step score is strictly greater than zero. For each PEFT technique, the direction-specific zero-shot denominator is 84 cases and the overall denominator is 168 cases.

C.2.5 PEFT technique hyperparameters

The following settings are explicitly configured in the released implementation:

- **Houlsby-style Adapters:** bottleneck dimension = 64.
- **Compacter:** reduction factor = 16; hypercomplex division = 4.
- **LoRA:** rank $r = 8$; $\alpha = 16$; dropout = 0.0.

-
- **Prefix Tuning:** number of prefix tokens = 5; reparameterisation enabled; embedding dimension = 512; intermediate dimension = 512.

Other method-level defaults inherit the installed OpenDelta implementation.

C.2.6 Training budgets and optimisation settings

Training is implemented using the Hugging Face `Trainer`. Both pre-porting and post-porting phases use a learning rate of 1×10^{-4} , gradient accumulation of one, and warm-up over 10% of the corresponding maximum training steps. The learning-rate scheduler is not explicitly overridden and therefore inherits the Hugging Face `TrainingArguments` default used in the experimental environment. Pre-porting and post-porting budgets are summarised in Table C.1.

Batch sizes. For pre-porting training, the per-device example batch size is 32 for SST-2, Rotten Tomatoes, SICK, and MRPC, and 8 for MNLI, QQP, and SAMSum. For post-porting training, the corresponding sizes are 16 and 8. Because every task uses fixed maximum input lengths, these correspond to the following nominal input-token batch sizes:

- **Pre-porting:** Rotten Tomatoes, SST-2, MNLI, and SAMSum use 4,096 tokens; SICK uses 2,240; MRPC uses 2,560; QQP uses 2,800.
- **Post-porting:** MNLI, SAMSum, and QQP match their pre-porting batch sizes; SST-2 and Rotten Tomatoes use 2,048 tokens; SICK uses 1,120; MRPC uses 1,280.

Optimiser defaults and model selection. The implementation does not explicitly pass an optimiser backend, weight decay, gradient-clipping threshold, or mixed-precision option to `TrainingArguments`; these settings therefore inherit the defaults of the installed Transformers version. Gradient checkpointing is not enabled, and no early-stopping callback is used. Dataset split usage follows the full-study protocol described in Appendix C.2.2: where labelled test sets are unavailable, 10%

of the original training split is reserved for development-time evaluation and the original development split is held out for final evaluation; otherwise, the provided development and test splits serve those respective roles.

C.2.7 Computational resources

The PEFT-Port experiments were run on NVIDIA A100 GPUs with 80GB memory, using one GPU per task configuration. The chapter reports controlled learning-step budgets rather than wall-clock time or GPU-hour totals, because the central comparisons concern matched pre-porting and post-porting adaptation budgets rather than hardware-specific runtime.

Appendix D

Supplementary Material for Chapter 6 (Budgeted LoRA)

Appendix roadmap. This appendix provides supplementary material for Chapter 6. Sections D.1.1–D.1.3 report the ablation studies underlying student construction, LoRA configuration, and budget scheduling/compression choices. Section D.2 expands the student-side training-compute proxy introduced in Section 6.5.1. Sections D.3.1 and D.3.2 document the full ICL protocol and per-task results underlying Section 6.5.2.

D.1 Ablation studies

This section supports the method choices described in Section 6.3 and the compared-method setup in Section 6.4.4. It reports the ablations used to fix the student construction, KD hyperparameters, LoRA configuration, and Budgeted LoRA schedule before the main results are reported.

I conduct a structured ablation study to isolate the key design choices underlying the final Chapter 6 configuration. The ablations are organised progressively: I first establish the distillation baseline and student-construction regime, then introduce LoRA-specific choices, and finally analyse the scheduling and compression components specific to Budgeted LoRA.

For most ablation experiments, I use 1/40 of the full distillation token budget, since broad hyperparameter exploration at the full 2.55B-token scale would be prohibitively expensive. The main exception is budget-scheduler tuning for Budgeted LoRA, where I use the full token budget. Scheduler behaviour is strongly token-dependent, and I did not establish a reliable transfer law from short runs to full-budget training; tuning it directly at the target budget is therefore more faithful than using a reduced-budget proxy.

All ablations use the same optimisation setup and held-out validation signal as the main experiments, so that model-selection decisions are guided by the same evaluation criterion used in the chapter.

D.1.1 Distillation hyperparameters and student initialisation

Distillation objective and varied factors. The ablations in this subsection use the distillation objective defined in Section 6.3.1. Here I vary only the student-construction regime and the KD hyperparameters used with that objective. Concretely, I study the effect of the distillation temperature τ_{KD} , the KD mixing weight λ_{KD} , and the student initialisation strategy.

All experiments in this stage use a 0.13B teacher and a 6-layer student constructed first via truncation (taking the first six layers), before later comparing alternative student-construction modes.

I explore the following search space:

- Initialisation: `warmstart`, `scratch`
- Temperature: $T \in \{1.0, 2.0, 3.0, 4.0\}$
- Distillation weight: $\lambda \in \{0.3, 0.5, 0.7, 0.9\}$

I find that warm-starting the student from a truncated checkpoint consistently outperforms training from scratch, and the best distillation configuration is $T=2.0$, $\lambda=0.5$ (Table D.1), which I use as a starting point for further exploration.

Temperature T	λ_{KD}	Warmstart PPL ↓	Scratch PPL ↓
1.0	0.3	34.96	53.01
2.0	0.5	34.08	52.61
3.0	0.7	34.50	62.40
4.0	0.9	37.39	85.08

Table D.1: The Pile Validation perplexity (PPL) for different distillation hyperparameter settings, comparing warmstart and scratch initialisation. Lower is better.

Student construction mode	PPL ↓
truncated	34.08
mixed	28.13
middle	29.58
last	32.15

Table D.2: The Pile Validation perplexity (PPL) for different student build modes under warmstart KD Full with fixed distillation hyperparameters ($T=2.0$, $\lambda_{\text{KD}}=0.5$). Lower is better.

Student construction. I next refine the warm-start strategy by varying how layers are selected from the teacher. I compare four variants: *first*, *middle*, *last*, and *mixed*. The first, middle, and last variants select contiguous blocks from the corresponding depth regions of the teacher. The mixed strategy instead anchors the student at the first and last teacher layers, then fills the remaining slots with approximately evenly spaced interior layers across depth. This provides coverage of early, middle, and late representations while preserving both low-level and high-level processing stages. Empirically, the mixed strategy consistently yields the best performance, suggesting that a depth-diverse initialisation is more effective than copying a single contiguous block (Table D.2).

I then perform a second sweep over (T, λ) for the top student constructions (*mixed* and *middle*), using:

$$T \in \{1.0, 2.0, 3.0\}, \quad \lambda \in \{0.3, 0.5, 0.8\}.$$

The best configuration is $T=3.0$, $\lambda=0.8$ with the mixed initialisation (Table D.3). This configuration is fixed for all subsequent experiments.

Temperature T	λ_{KD}	Mixed PPL $\downarrow$	Middle PPL $\downarrow$
1.0	0.3	29.01	30.55
2.0	0.5	28.11	29.58
3.0	0.8	27.84	29.92

Table D.3: The Pile Validation perplexity (PPL) for warmstart KD Full under two student build modes, mixed and middle, across different distillation hyperparameter settings. Lower is better.

LoRA rank r	LoRA α	PPL $\downarrow$
16	32	39.65
32	64	35.97
64	128	33.20
128	256	31.13

Table D.4: The Pile Validation perplexity (PPL) for warmstart KD LoRA across different LoRA ranks under fixed distillation settings ($T=3.0$, $\lambda_{\text{KD}}=0.8$) and mixed student build mode. Lower is better.

D.1.2 LoRA design choices

This subsection supports the KD LoRA baseline and the shared LoRA settings used by Budgeted LoRA in Sections 6.3.2 and 6.4.4.

I next introduce LoRA and study its additional design choices. I attach LoRA modules to all linear layers in the student (7 per layer across 6 layers, totalling 42 modules: `q_proj`, `k_proj`, `v_proj`, `o_proj`, `down_proj`, `gate_proj`, `up_proj`).

I sweep the LoRA rank:

$$r \in \{16, 32, 64, 128\}.$$

The best-performing configuration uses rank $r=128$ (Table D.4), which I fix for all LoRA-based experiments.

I also sweep over multiple student construction modes (*truncated*, *mixed*, *middle*, and *last*) to test whether LoRA prefers a different student build than full KD, given that LoRA is inserted across multiple frozen modules (cf. the structural view of LoRA in the PEFT-Ref framework Sabry and Belz (2023)).

The same qualitative pattern holds as in full KD: *mixed* remains the best-

performing construction (31.14 PPL), followed by *middle* (41.52), *last* (44.72), and *truncated* (48.09). Thus, while performance is sensitive to student construction, the best choice among those tested is unchanged, and the KD Full hyperparameters also perform best in this LoRA sweep.

D.1.3 Budgeted LoRA: scheduling and compression

This subsection supports the budget schedule, dense-removal threshold, and compression settings used in Sections 6.3.3 and 6.3.4.

Finally, I analyse the components specific to Budgeted LoRA.

Budget scheduling. I focus on the most aggressive budget profile ($F=0.0$) and vary the decay schedule controlling when dense computation is removed during training. I consider:

- No scheduling: (0.0) — immediate application of the final budget
- Short decay: (0.1, 0.3) — early transition with strong compute savings
- Long decay: (0.1, 0.9) — gradual transition with weaker savings

The short decay provides the lowest perplexity among the tested schedules while retaining stronger proxy savings than the long decay (Table D.5). Immediate dense removal does not maximise final LoRA usage: under no scheduling, the average retained rank is only 120. While one might expect this setting to push all computation into the low-rank pathway, in practice the transition is too abrupt and results in worse optimisation, with a lower final effective rank than both short decay (128) and long decay (124). This pattern suggests that the schedule governs the handoff of usable capacity from dense to low-rank computation while also supporting optimisation stability. In this setting, short decay provides enough adaptation time for this transfer while still enforcing strong compute savings.

Schedule	PPL↓	Train compute ↓
No schedule	35.7	0.25×
Short decay (0.1–0.3)	25.1	0.38×
Long decay (0.1–0.9)	34.8	0.58×

Table D.5: Effect of budget scheduling ($F=0.0$).

Dense removal and compression thresholds. I fix the dense-removal threshold during training and post-training compression to 10^{-3} : dense components with retention values below this numerical near-zero tolerance are skipped during training and removed during post-training compression.

For post-training compression, I sweep thresholds for:

- LoRA rank gating: $\{0.3, 0.5, 0.9\}$
- Dense SVD threshold: $\{0.5, 0.7, 0.9\}$

At compression time, dense components with near-zero retention are removed, components with high retention are kept and merged with LoRA, and intermediate cases are approximated by a low-rank SVD of the residual dense contribution. Specifically, if a dense matrix W has learned retention coefficient α , I approximate αW with a rank- k factorisation, where k is chosen adaptively up to a fixed maximum rank budget (128 ranks). This provides a middle ground between complete dense removal and full dense retention. While the global budget determines how much dense computation is retained overall, rank gating acts as a secondary optimisation that removes inactive low-rank directions, allowing the model to further compress adaptation capacity where it is not needed.

A rank threshold of 0.3 and a dense-SVD threshold of 0.7 provide the best validation performance among the tested combinations across $F \in \{0.0, 0.4\}$.

Summary. Across these ablations, the selected distillation hyperparameters transfer from KD Full to the LoRA-based runs, while schedule and compression choices materially affect Budgeted LoRA. This supports fixing the shared hyperparameters within the reported comparison and focusing the remaining sweep on the budget

mechanism.

D.2 Training compute estimation

This section supports the student-side efficiency quantities reported in Section 6.4.2 and Table 6.2. I expand the student-side training-compute proxy introduced in Section 6.5.1. The goal is to support the relative comparisons in Table 6.2 by deriving a simple proxy for KD Full, KD LoRA, and Budgeted LoRA at the level of adapted modules, rather than exact end-to-end FLOPs.

D.2.1 Training compute model

Consider a dense linear layer with forward compute cost D . I use a first-order approximation in which forward and backward passes incur comparable cost.

KD Full. When fine-tuning all parameters, training incurs:

- forward pass through dense weights: D ,
- backward pass to activations: D ,
- backward pass to dense weights: D .

This yields the proxy:

$$C_{\text{full}} \approx 3D. \tag{D.1}$$

KD LoRA. In KD LoRA, dense weights are frozen. The computation still includes:

- forward pass through dense base: D ,
- backward pass to activations: D ,

but *not* backward computation for dense weights. In addition, the LoRA path introduces trainable low-rank modules with forward cost L , which incur full forward

and backward cost. This gives:

$$C_{\text{loa}} \approx 2D + 3L. \quad (\text{D.2})$$

Thus, KD LoRA is expected to be cheaper than full fine-tuning unless the low-rank cost L is disproportionately large.

Budgeted LoRA. Budgeted LoRA follows the same structure as KD LoRA, but progressively removes the dense path during training. Let $\bar{d} \in [0, 1]$ denote the *average retained dense fraction* over the full training run. The compute becomes:

$$C_{\text{bud}} \approx 2\bar{d}D + 3L. \quad (\text{D.3})$$

D.2.2 Average dense retention

In this formulation, the budget acts as an upper bound on retained dense compute: the controller greedily lowers module-level retention coefficients until the current dense cost satisfies the scheduled target.

I use a budget schedule that begins at 10% of training, reaches the final dense fraction F by 30%, and remains constant thereafter. Approximating the transition as linear, the average retained dense fraction is:

$$\bar{d} \approx 0.1 \cdot 1 + 0.2 \cdot \frac{1 + F}{2} + 0.7 \cdot F = 0.2 + 0.8F. \quad (\text{D.4})$$

Thus:

$$F = 0.0 \Rightarrow \bar{d} = 0.20, \quad (\text{D.5})$$

$$F = 0.4 \Rightarrow \bar{d} = 0.52. \quad (\text{D.6})$$

D.2.3 Instantiated values

From the compression summaries over replaced modules, I obtain:

$$D = 5.131 \times 10^7, \tag{D.7}$$

$$D + L = 6.400 \times 10^7 \quad \Rightarrow \quad L = 1.269 \times 10^7. \tag{D.8}$$

Using these values:

KD Full.

$$C_{\text{full}} \approx 3D = 1.5393 \times 10^8. \tag{D.9}$$

KD LoRA.

$$C_{\text{loa}} \approx 2D + 3L = 1.4069 \times 10^8, \tag{D.10}$$

which corresponds to:

$$\frac{C_{\text{loa}}}{C_{\text{full}}} \approx 0.91. \tag{D.11}$$

Budgeted LoRA ($F = 0.0$).

$$C_{\text{bud},0.0} \approx 2(0.20)D + 3L = 5.8594 \times 10^7, \tag{D.12}$$

$$\frac{C_{\text{bud},0.0}}{C_{\text{full}}} \approx 0.38. \tag{D.13}$$

Budgeted LoRA ($F = 0.4$).

$$C_{\text{bud},0.4} \approx 2(0.52)D + 3L = 9.1432 \times 10^7, \tag{D.14}$$

$$\frac{C_{\text{bud},0.4}}{C_{\text{full}}} \approx 0.59. \tag{D.15}$$

D.2.4 Summary

Under this training-compute proxy, the methods follow the ordering:

$$\text{Budgeted LoRA } (F=0.0) < \text{Budgeted LoRA } (F=0.4) < \text{KD LoRA} < \text{KD Full}. \quad (\text{D.16})$$

This reflects that KD LoRA removes dense weight-gradient computation, while Budgeted LoRA further reduces cost by progressively eliminating dense paths during training.

Relation to 6ND. A common first-order estimate for dense language-model training is $6ND$ FLOPs, where N is the number of model parameters and D is the number of training tokens; the factor counts forward and backward computation per parameter and token (Kaplan et al., 2020; Hoffmann et al., 2022). Expressed in multiply-accumulate operations, this is approximately $3ND$, because one MAC is conventionally counted as two FLOPs (Qiu et al., 2024). I use this MAC-style view at the adapted-module level: full dense training pays one forward, one activation-backward, and one weight-backward term, giving $C_{\text{full}} \approx 3D$ for a dense layer of forward cost D .

Since all methods are trained on the same number of tokens, token count is a common multiplicative factor and cancels in relative comparisons.

D.3 In-context learning capabilities

This section supports the ICL-retention results in Section 6.5.2. It provides the full probe inventory, prompt format, and task-level results underlying the macro-averaged ICL composite reported in the main chapter.

D.3.1 Function-style probes

This section documents the full ICL protocol underlying Section 6.5.2. The 19 function-style tasks are adapted from Todd et al. (2024). The main text summarises

the task families, prompting protocol, and output extraction rules; here I provide the concrete prompt format, the complete task inventory, and the task-level evaluation details.

Prompting controls. I fix the prompt template across models and average results across multiple demonstration seeds, so that variation reflects the sampled demonstrations rather than changes in prompt wording. Following the default evaluation template, each in-context example is formatted as `Q:<input>` followed by `A:<output>`, with a newline between input and output and a blank line between demonstrations. No additional natural-language instruction prefix is used. The query is appended in the same format and terminates at `A:`, requiring the model to generate the target continuation. For example, for a capitalization-style task, a 3-shot prompt would be:

`Q:dog`

`A:DOG`

`Q:house`

`A:HOUSE`

`Q:train`

`A:TRAIN`

`Q:river`

`A:`

where the desired continuation is `RIVER`.

The task suite spans several broad categories:

Table D.6: Tasks, evaluation metrics, and shot counts used to compute the ICL composite in Section 6.5.2. Here HITS@1 accuracy corresponds to exact-match correctness of the first generated answer after the task-level canonicalisation used by the evaluation script.

Tasks	Metric	Shots	Notes
capitalize	HITS@1 Acc	10	Convert the entire input string to uppercase (e.g., “hello” → “HELLO”).
capitalize_first_letter	HITS@1 Acc	10	Uppercase the first character only; leave the rest unchanged (“alpha” → “Alpha”).
capitalize_last_letter	HITS@1 Acc	10	Uppercase the final character only (“gamma” → “gammaA”).
lowercase_first_letter	HITS@1 Acc	10	Lowercase the first character only (“Alpha” → “alpha”).
lowercase_last_letter	HITS@1 Acc	10	Lowercase the final character only (“GammaA” → “Gamma”).
next_capital_letter	HITS@1 Acc	10	Map an uppercase letter to its successor in the alphabet (e.g., A→B; wraparound optional).
next_item	HITS@1 Acc	10	Given an item from an ordered category (day, month, letter, number word), output the next item (“Monday” → “Tuesday”).
prev_item	HITS@1 Acc	10	As above, but return the previous item (“Tuesday” → “Monday”; wraparound for cyclic lists).
word_length	HITS@1 Acc	10	Return the number of characters in the input word (“token” → 5).
alphabetically_first_3	HITS@1 Acc	10	From a list of 3 strings, choose the alphabetically earliest.
alphabetically_first_5	HITS@1 Acc	10	From a list of 5 strings, choose the alphabetically earliest.

Continued on next page

Tasks	Metric	Shots	Notes
alphabetically_last_3	HITS@1 Acc	10	From a list of 3 strings, choose the alphabetically latest.
alphabetically_last_5	HITS@1 Acc	10	From a list of 5 strings, choose the alphabetically latest.
choose_first_of_3	HITS@1 Acc	10	From a list of 3 items, select the first item by position.
choose_first_of_5	HITS@1 Acc	10	From a list of 5 items, select the first item by position.
choose_last_of_3	HITS@1 Acc	10	From a list of 3 items, select the last item by position.
choose_last_of_5	HITS@1 Acc	10	From a list of 5 items, select the last item by position.
choose_middle_of_3	HITS@1 Acc	10	From a list of 3 items, select the middle item by position.
choose_middle_of_5	HITS@1 Acc	10	From a list of 5 items, select the middle item by position.

D.3.2 Performance

This section provides the task-level results underlying Section 6.5.2. Table D.7 gives the per-task breakdown behind Figure 6.2. KD Full’s mean scores increase with teacher scale, KD LoRA’s aggregate decreases, and the budgeted variants have higher means than KD LoRA on many tasks at the larger teacher scales.

The positive differences occur across several task families, including selection, transformation, and sequence probes, but not on every task. They remain descriptive results from the fixed warm-started student setup.

	Full Model		KD Full			KD LoRA		
	0.13B	0.13B	0.5B	1B	0.13B	0.5B	1B	
alphabetically_first-3	4.9 ± 0.6	4.0 ± 0.6	9.8 ± 0.5	11.2 ± 1.2	3.3 ± 0.1	1.4 ± 0.3	3.2 ± 0.2	
alphabetically_first-5	4.0 ± 0.6	5.1 ± 0.6	8.3 ± 0.1	8.8 ± 0.6	3.8 ± 0.3	3.3 ± 0.5	4.1 ± 0.4	
alphabetically_last-3	3.4 ± 0.5	2.8 ± 0.4	9.6 ± 1.6	11.7 ± 0.4	2.2 ± 0.4	2.4 ± 0.2	4.2 ± 0.1	
alphabetically_last-5	2.5 ± 0.7	2.4 ± 0.5	5.7 ± 1.0	6.2 ± 0.4	2.5 ± 0.2	3.0 ± 0.3	3.8 ± 1.1	
capitalize	8.0 ± 1.0	4.4 ± 1.2	24.7 ± 1.4	36.1 ± 1.9	3.2 ± 0.9	2.3 ± 0.9	3.3 ± 0.8	
capitalize_first_letter	10.1 ± 1.2	8.7 ± 0.5	20.2 ± 2.0	27.1 ± 0.4	7.7 ± 0.6	5.1 ± 0.7	1.9 ± 0.5	
capitalize_last_letter	4.8 ± 1.3	7.0 ± 1.1	8.6 ± 1.1	6.4 ± 1.2	8.7 ± 1.4	5.8 ± 0.5	3.8 ± 0.4	
choose_first-of-3	10.3 ± 2.3	5.8 ± 0.5	27.2 ± 0.6	33.3 ± 1.4	3.2 ± 0.6	2.0 ± 0.3	3.0 ± 0.6	
choose_first-of-5	8.7 ± 1.3	4.6 ± 0.5	22.8 ± 0.9	25.6 ± 0.7	3.3 ± 0.2	1.8 ± 0.4	3.7 ± 1.2	
choose_last-of-3	2.0 ± 0.4	1.6 ± 0.1	2.8 ± 0.7	5.4 ± 1.0	1.6 ± 0.6	1.3 ± 0.3	1.8 ± 0.3	
choose_last-of-5	1.5 ± 0.3	2.0 ± 0.5	2.3 ± 0.3	3.8 ± 0.4	1.1 ± 0.3	1.2 ± 0.1	1.8 ± 0.7	
choose_middle-of-3	1.7 ± 0.6	1.7 ± 0.5	2.6 ± 0.5	3.8 ± 0.1	1.5 ± 0.6	1.2 ± 0.3	1.3 ± 0.3	
choose_middle-of-5	1.6 ± 0.3	1.3 ± 0.7	2.0 ± 0.7	2.6 ± 0.5	1.3 ± 0.5	1.7 ± 0.4	1.2 ± 0.6	
lowercase_first_letter	5.5 ± 0.8	6.3 ± 0.8	12.2 ± 1.1	15.7 ± 1.2	5.3 ± 1.1	0.8 ± 0.4	0.0 ± 0.0	
lowercase_last_letter	11.1 ± 0.8	10.4 ± 0.5	11.0 ± 1.2	12.5 ± 1.7	13.1 ± 1.3	1.9 ± 0.2	0.2 ± 0.2	
next_capital_letter	4.9 ± 1.1	5.6 ± 1.1	4.6 ± 1.2	3.7 ± 0.9	4.5 ± 0.7	5.2 ± 0.2	1.8 ± 0.4	
next_item	3.4 ± 2.2	3.8 ± 1.3	5.9 ± 2.2	7.8 ± 2.4	2.7 ± 1.0	1.3 ± 0.6	0.6 ± 0.6	
prev_item	2.5 ± 0.8	4.4 ± 0.0	5.9 ± 1.0	5.3 ± 1.3	1.5 ± 1.0	1.5 ± 0.4	1.5 ± 0.7	
word_length	9.9 ± 1.0	10.1 ± 1.0	10.7 ± 0.9	9.1 ± 0.5	12.0 ± 1.1	10.1 ± 1.3	9.3 ± 0.5	
ICL composite (macro) ↑	5.3 ± 0.9	4.8 ± 0.7	10.4 ± 1.0	12.4 ± 1.0	4.3 ± 0.7	2.8 ± 0.4	2.7 ± 0.5	

Table D.7: Per-task in-context learning (ICL) performance on 19 Todd et al. (2024) function-style tasks. I report mean $\pm$ standard deviation across runs. Columns group methods (Full model, KD Full, KD LoRA, and Budgeted LoRA) across teacher scales (0.13B, 0.5B, 1B). The final row shows the macro-averaged ICL composite score used in Figure 6.2. At the larger teacher scales, the budgeted variants have higher means than KD LoRA on many, but not all, tasks; this comparison is specific to the reported setup.

	Bud. LoRA (F=0.0)			Bud. LoRA (F=0.4)		
	0.13B	0.5B	1B	0.13B	0.5B	1B
alphabetically_first_3	2.2 ± 0.5	4.1 ± 0.4	4.2 ± 0.2	3.4 ± 0.7	3.3 ± 0.5	3.7 ± 0.2
alphabetically_first_5	1.8 ± 0.5	3.5 ± 0.4	4.8 ± 1.0	3.9 ± 0.8	3.9 ± 0.5	3.7 ± 0.7
alphabetically_last_3	1.9 ± 0.5	3.9 ± 1.0	3.5 ± 1.2	1.4 ± 0.1	2.9 ± 0.1	4.1 ± 1.0
alphabetically_last_5	1.0 ± 0.2	3.4 ± 0.3	2.9 ± 0.5	1.5 ± 0.2	3.3 ± 0.6	3.3 ± 0.3
capitalize	4.2 ± 0.9	3.8 ± 0.5	4.9 ± 0.5	3.0 ± 0.7	3.4 ± 1.1	4.9 ± 0.9
capitalize_first_letter	7.1 ± 0.8	7.2 ± 1.1	7.4 ± 1.1	7.2 ± 0.7	7.1 ± 0.4	9.6 ± 0.4
capitalize_last_letter	7.3 ± 1.7	7.5 ± 1.3	8.9 ± 1.2	8.8 ± 1.1	12.5 ± 0.6	11.3 ± 0.2
choose_first_of_3	1.7 ± 0.0	6.0 ± 0.5	4.1 ± 0.5	2.6 ± 0.8	3.2 ± 0.5	5.9 ± 0.4
choose_first_of_5	1.6 ± 0.1	4.2 ± 0.3	3.4 ± 0.4	2.8 ± 0.7	2.8 ± 0.9	4.2 ± 0.6
choose_last_of_3	1.8 ± 0.4	1.5 ± 0.6	1.2 ± 0.6	1.2 ± 0.5	1.5 ± 0.3	1.4 ± 0.2
choose_last_of_5	1.3 ± 0.4	1.6 ± 0.3	1.7 ± 0.0	1.2 ± 0.3	1.7 ± 0.3	1.7 ± 0.4
choose_middle_of_3	2.3 ± 0.3	1.6 ± 0.6	1.8 ± 0.5	1.2 ± 0.6	1.6 ± 0.5	1.5 ± 0.3
choose_middle_of_5	2.0 ± 0.3	1.5 ± 0.4	2.1 ± 0.5	1.1 ± 0.6	1.4 ± 0.1	2.1 ± 1.0
lowercase_first_letter	5.4 ± 0.6	0.5 ± 0.2	5.3 ± 0.6	5.6 ± 0.7	4.5 ± 1.1	8.1 ± 0.9
lowercase_last_letter	8.1 ± 0.3	0.2 ± 0.1	11.3 ± 0.4	10.3 ± 0.3	9.2 ± 0.7	11.0 ± 0.5
next_capital_letter	5.4 ± 0.3	5.7 ± 0.5	4.6 ± 0.4	4.5 ± 1.3	5.3 ± 0.8	4.7 ± 1.1
next_item	2.7 ± 0.4	2.7 ± 2.2	4.2 ± 0.7	2.7 ± 0.7	2.5 ± 1.7	8.0 ± 1.0
prev_item	3.0 ± 0.7	3.0 ± 1.0	4.6 ± 1.8	1.9 ± 1.3	2.7 ± 1.5	6.8 ± 2.0
word_length	11.2 ± 1.2	14.9 ± 0.3	13.7 ± 0.7	9.6 ± 0.5	11.1 ± 1.1	10.5 ± 0.3
ICL composite (macro) ↑	3.8 ± 0.5	4.0 ± 0.6	5.0 ± 0.7	3.9 ± 0.7	4.4 ± 0.7	5.6 ± 0.7

Table D.7 (continued): Budgeted LoRA per-task results.

Appendix E

Supplementary Material for Chapter 7 (In-Context Learning Capability)

E.1 Appendix roadmap

This appendix collects supplementary material for Chapter 7. The main chapter defines the core concepts, reported metrics, and causal tests needed to interpret the results. This appendix provides extended benchmark details, robustness checks, per-task tables, and supplementary telemetry analyses.

- Section E.2 lists the benchmark tasks, metrics, shot counts, and supplementary prompting details used in the ICL evaluations.
- Section E.3 reports span-length and mix-ratio sweeps that support the main Bi-Induct design choices.
- Section E.2.2 provides supplementary copy-head telemetry and mix-ratio analyses beyond the main-text figures.
- Section E.4 provides full per-task capability results and additional robustness analyses.

- Section E.5 reports the full supplementary head-ablation plots and per-task tables.

E.2 Supplementary evaluation details

The main operational definitions and four-axis evaluation protocol are given in Sections 7.1.1 and 7.2.4. This appendix keeps the extended benchmark list, prompting controls, and robustness details so that the main chapter remains self-contained without becoming overloaded.

E.2.1 Benchmarks and protocols

Table E.1: Benchmarks, evaluation metrics, and shot counts used to compute the behavioural ICL composite in Section 7.3.1.

Benchmark / Tasks	Metric	Shots	Notes
MMLU (Hendrycks et al., 2021)	Acc	3	57 subject areas; standard 3-shot setup.
Winogrande (Sakaguchi et al., 2019)	Acc	3	Commonsense coreference.
CommonSenseQA (Talmor et al., 2019)	Acc	3	Multiple choice commonsense.
PIQA (Bisk et al., 2020)	Acc	3	Physical commonsense.
HellaSwag (Zellers et al., 2019)	Acc	3	Story completion.
TriviaQA-Wiki (Joshi et al., 2017)	EM	3	Open-domain QA, Wikipedia evidence.
BBH (CoT) (Suzgun et al., 2022)	EM	3	Few hard tasks with chain-of-thought prompts.

Continued on next page

Benchmark / Tasks	Metric	Shots	Notes
OpenBookQA (Mihaylov et al., 2018)	Acc	3	Elementary science QA.
ARC-Challenge (Clark et al., 2018)	Acc	3	Difficult science questions.
GPQA (Rein et al., 2023)	Acc	3	Graduate-level QA.
GSM-8K (Cobbe et al., 2021)	EM	3	Math word problems with short CoT.
MathQA (Amini et al., 2019)	Acc	3	Programmatic math QA.
BoolQ (Clark et al., 2019)	Acc	3	Yes/No reading comprehension.
LAMBADA (OpenAI) (Paperno et al., 2016)	Acc	3	Cloze final-word prediction.

Metric abbreviations. Acc denotes accuracy, EM denotes exact match, and CoT denotes chain-of-thought prompting.

Aggregation. The main text reports macro ICL composites and per-task scores. Each composite is an unweighted mean over the selected tasks within that evaluation family. All figures and tables report means over seeds unless otherwise stated. The full list of standard language-model benchmarks is given in Table E.1.

Prompting controls. For few-shot tasks, I fix the prompt template and average across demonstration seeds. For robustness, I randomise demonstration order and evaluate sensitivity to shot count and label permutation in Appendix E.4.2.

E.2.2 Supplementary copy-head telemetry details

The main text defines induction signatures, anti-induction signatures, copy-score computation, and the top-2% head-localisation rule in Section 7.1.1. Here I keep

the supplementary implementation details and concentration summaries used for the appendix telemetry analyses.

Copy telemetry is computed on 50,000 fresh copy probes disjoint from training. Each probe has the form $x = s \langle \text{sep} \rangle s$, with a uniformly sampled token span s of length $L=500$. For reproducibility, this longer span differs from the $L=20$ span used in the main Bi-Induct curriculum and reduces the chance that telemetry simply reflects memorisation of the replacement-based training-intervention format.

In addition to the best-head score plotted in the main text, I report layerwise mass share:

$$\text{MassShare}_\ell^{(S)} = \frac{\sum_{h \in \text{Top}_\ell(S)} S(\ell, h)}{\sum_{h \in H_\ell} S(\ell, h)},$$

where $S \in \{\text{Score}_I, \text{Score}_A\}$ and $\text{Top}_\ell(S)$ denotes the top-2% heads in layer ℓ , with a one-head floor. Larger values indicate stronger concentration of copy-score mass in a small number of heads. The corresponding induction and anti-induction score definitions are not repeated here to avoid redefining the chapter’s core quantities; see Section 7.1.1.

E.2.3 Supplementary language-modelling quality check

The main text defines next-token cross-entropy and perplexity (PPL) in Section 7.2.4. For all PPL evaluations in Chapter 7, I use the same fixed 0.2% THE PILE validation slice, selected by stable hash partition, with identical tokenisation and evaluation settings across runs. Under the iso-FLOPs comparisons used in the chapter, this held-out PPL measurement serves as a language-modelling quality check rather than as a separate optimisation target.

E.3 Design ablations and robustness checks

The design-lab sweeps use an earlier 5-shot diagnostic over five standard language-model benchmarks to select the operating point efficiently. The main cross-scale evaluation instead uses the broader 3-shot protocol over 14 standard benchmarks,

together with the 10-shot function-probe suite, as specified in Section 7.2.4.

E.3.1 Span-length sweep

This subsection reports the span-length sweep used to choose the default Bi-Induct span length. At **0.13B**, with a fixed initial mix of 25% linearly annealed over the full token budget, I sweep $L \in \{5, 20, 500\}$ and report two endpoints of practical interest—(i) a 5-shot ICL composite over five standard LM benchmarks and (ii) held-out LM perplexity (PPL). I defer the function-probe suite of Todd et al. (2024) to the cross-scale experiments, where relative differences are more interpretable and the added compute is justified; the span-length results for this subsection are summarised in Table E.2.

Across curricula, $L=20$ is a stable operating point that balances ICL and calibration: for *Induction*, 31.9 ICL / 23.9 PPL (vs. 30.7/23.8 at $L=5$ and 31.8/24.0 at $L=500$); for *Anti*, 32.1/24.0 (vs. 31.6/23.8 at $L=5$, 31.7/24.5 at $L=500$) respectively; for *Balanced*, 31.2/24.0 (vs. 31.4/23.8 at $L=5$, 32.0/24.0 at $L=500$). Very short spans ($L=5$) underperform on the ICL composite, while very long spans ($L=500$) offer no consistent ICL gain and tend to slightly worsen PPL. Hence I adopt $L=20$ for the remaining experiments. Beyond the ICL / PPL balance, shorter spans are operationally attractive: they minimise snippet length $2L+|\text{SEP}|$, which reduces potential overhead and makes it easier to *pack* snippets alongside natural sequences to exploit variable-length kernels, yielding compute savings when such packing is enabled.¹

E.3.2 Initial mix-ratio sweep

For the mix-ratio sweep, I fixed the anneal to span the *full* training budget (2.6B tokens, following the Chinchilla parameter–token rule of thumb (Hoffmann et al., 2022)), held the span length at $L=20$, and swept the initial mix ratio over $\{25\%$,

¹All reported results are at iso-FLOPs; I do *not* take packing credits in my comparisons. Packing is a deployment optimisation, not part of the evaluation protocol.

	Baseline		Induction		
	–	5	20	500	
MMLU $\uparrow$	25.2 $\pm$ 0.3	25.3 $\pm$ 0.5	25.1 $\pm$ 0.4	25.3 $\pm$ 0.5	
ARC-Challenge $\uparrow$	18.1 $\pm$ 0.6	18.7 $\pm$ 0.4	18.6 $\pm$ 1.3	18.1 $\pm$ 0.6	
BoolQ $\uparrow$	52.8 $\pm$ 4.6	46.2 $\pm$ 5.2	53.5 $\pm$ 5.9	53.4 $\pm$ 3.7	
LAMBADA $\uparrow$	7.6 $\pm$ 0.7	6.7 $\pm$ 0.6	6.4 $\pm$ 0.6	6.8 $\pm$ 0.6	
PIQA $\uparrow$	56.6 $\pm$ 0.5	56.5 $\pm$ 0.6	55.8 $\pm$ 0.5	55.5 $\pm$ 0.2	
ICL composite (macro) $\uparrow$	32.1 $\pm$ 0.9	30.7 $\pm$ 2.4	31.9 $\pm$ 1.1	31.8 $\pm$ 1.7	
PPL $\downarrow$	21.8	23.8	23.9	24.0	

Table E.2: Span-length sweep at **0.13B** on THE PILE. All curricula are linearly annealed over the full training budget with initial mix of 25%. Results are averaged over six seeds. Evaluation is 5-shot. I report per-task accuracies, the macro ICL composite, and held-out perplexity (PPL).

	Anti-induction			Balanced		
	5	20	500	5	20	500
MMLU $\uparrow$	25.3 ± 0.4	25.1 ± 0.5	24.8 ± 0.5	25.2 ± 0.4	24.8 ± 0.3	25.1 ± 0.2
ARC-Challenge $\uparrow$	18.5 ± 0.7	18.1 ± 0.4	17.6 ± 0.6	18.6 ± 0.2	18.5 ± 0.6	17.6 ± 0.4
BoolQ $\uparrow$	50.8 ± 6.8	54.5 ± 1.4	53.4 ± 2.9	51.2 ± 2.5	50.1 ± 3.5	54.7 ± 4.9
LAMBADA $\uparrow$	6.4 ± 0.5	6.7 ± 0.5	6.2 ± 0.2	7.0 ± 0.1	6.9 ± 0.4	6.5 ± 0.5
PIQA $\uparrow$	55.9 ± 0.3	55.9 ± 0.6	56.4 ± 0.8	56.0 ± 0.6	55.8 ± 0.5	56.0 ± 0.5
ICL composite (macro) $\uparrow$	31.6 ± 1.2	32.1 ± 0.3	31.7 ± 1.4	31.4 ± 3.1	31.2 ± 0.7	32.0 ± 2.2
PPL $\downarrow$	23.8	24.0	24.5	23.8	24.0	24.2

Table E.2 (continued): Anti-induction and Balanced results.

50%, 100% }. Table E.3 reports full per-task results.

Mechanistic readout: Figure E.1 summarises layerwise copy-head activity across mix ratios. I quantify the telemetry in three complementary ways: (i) *non-zero signature spread*, defined as the number of selected per-layer heads whose measured induction score is greater than zero; (ii) the number of selected heads above the fixed specialisation threshold of 0.5; and (iii) *peak sharpness*, the maximum selected-head score in each condition. I also examine depth concentration, namely whether prominent peaks cluster within particular layer ranges.

Does the replacement-based intervention improve induction-head quality vs. baseline? By counts above the 0.5 threshold, yes up to moderate mixes. The baseline shows 2 specialised heads. With **Induction** snippets I observe {3, 3, 1} specialised heads at {25%, 50%, 100%} mixes, **Balanced** yields {4, 4, 1}, and **Anti** yields {3, 1, 2}. By peak sharpness, **Balanced-25%** attains the highest induction head, with Induction and Anti close behind; at $\geq 50\%$ mixes, Balanced and Baseline retain similar peaks while Induction then Anti trail.

Does the curriculum create anti-induction heads? No. Even under Anti mixes, anti-induction scores remain far below the specialisation threshold; the best peaks are ≈ 0.01 (at 50% Anti and in Baseline), indicating no robust anti-induction circuit emerges.

How does mix ratio affect spread and depth concentration (induction)? For **Induction**, spread rises from 25% to 50% (3 $\rightarrow$ 5 heads) then contracts at 100% (2). For **Balanced**, spread is largest at 25% (5), then declines (4 at 50%, 2 at 100%). For **Anti**, induction heads peak at 25–50% (4 each) and drop to 2 at 100%. Depthwise, specialised heads shift deeper as mix increases: clusters are earlier at 25% (layers $\sim$ 2-4), mid-depth at 50% (layers $\sim$ 4-6), and later at 100% (around layer $\sim$ 8).

Takeaways: Baseline naturally forms a few strong induction heads. Replacing selected natural-sequence slots with synthetic snippets increases the number of strongly specialised induction heads at moderate mix ratios ($\leq 50\%$); higher mix

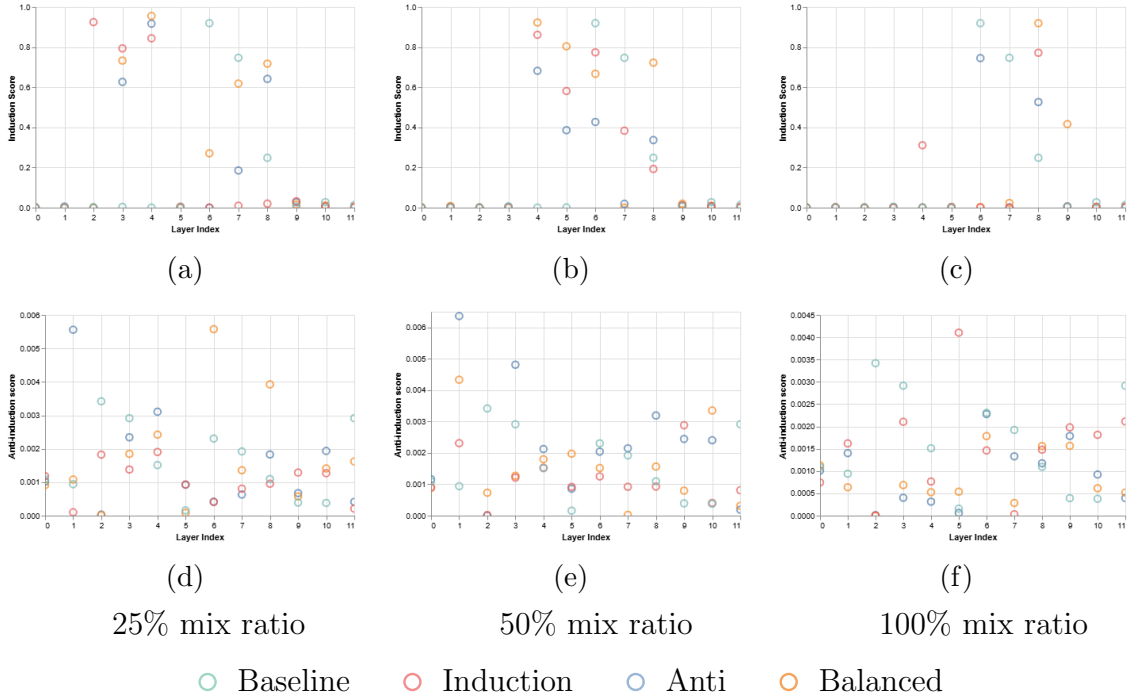


Figure E.1: Layer-wise copy-head telemetry. Top row: induction scores; bottom row: anti-induction scores. For each layer I plot the best-scoring head (top 2% by score with a floor of one head per layer), averaged over six seeds, for the 0.13B model with initial mix ratios: 25%, 50%, and 100%. Head counts for each model are given in Table 6.1.

ratios reduce their spread and shift prominent peaks towards deeper layers. None of the tested curricula, including Anti-Induction, produces a comparably strong anti-induction signature.

E.4 Additional results and analyses

E.4.1 In-context learning performance

The main text describes the task families, scoring procedures, and aggregate results in Section 7.3.1; this appendix reports the complete per-task results across scales for the standard language-model benchmarks and the function-style probes of Todd et al. (2024). For the standard benchmarks I use 3-shot evaluation; for the function-style probe suite I use 10-shot evaluation. Unless otherwise noted, metrics are accuracy (ACC), exact match (EM), or HITS@1 as appropriate, and all results are averaged over three seeds.

	Baseline		Induction		
	–	25%	50%	100%	
MMLU $\uparrow$	25.2 $\pm$ 0.26	25.1 $\pm$ 0.38	25.0 $\pm$ 0.33	24.9 $\pm$ 0.33	
ARC-Challenge $\uparrow$	18.1 $\pm$ 0.64	18.6 $\pm$ 1.28	18.1 $\pm$ 0.65	17.9 $\pm$ 0.89	
BoolQ $\uparrow$	52.8 $\pm$ 4.64	53.5 $\pm$ 5.92	49.6 $\pm$ 6.92	51.8 $\pm$ 4.51	
LAMBADA $\uparrow$	7.6 $\pm$ 0.69	6.4 $\pm$ 0.60	5.6 $\pm$ 0.50	4.7 $\pm$ 0.50	
PIQA $\uparrow$	56.6 $\pm$ 0.49	55.8 $\pm$ 0.48	55.4 $\pm$ 0.27	54.9 $\pm$ 0.49	
ICL composite (macro) $\uparrow$	32.06 $\pm$ 0.89	31.88 $\pm$ 1.08	30.76 $\pm$ 1.48	30.83 $\pm$ 0.98	
PPL $\downarrow$	21.8	23.9	26.0	31.4	

Table E.3: Initial mix-ratio sweep at **0.13B** on THE PILE. All curricula are linearly annealed over the full training budget with span fixed at $L = 20$. Results are averaged over six seeds. Evaluation is 5-shot. I report per-task accuracies, the macro ICL composite, and held-out perplexity (PPL).

	Anti-induction			Balanced		
	25%	50%	100%	25%	50%	100%
MMLU $\uparrow$	25.1 $\pm$ 0.48	25.2 $\pm$ 0.40	24.9 $\pm$ 0.41	24.8 $\pm$ 0.29	25.0 $\pm$ 0.27	24.6 $\pm$ 0.38
ARC-Challenge $\uparrow$	18.1 $\pm$ 0.42	18.0 $\pm$ 0.47	17.5 $\pm$ 0.91	18.5 $\pm$ 0.58	18.1 $\pm$ 0.44	17.8 $\pm$ 0.52
BoolQ $\uparrow$	54.5 $\pm$ 1.36	47.8 $\pm$ 6.78	51.1 $\pm$ 2.87	50.1 $\pm$ 3.52	46.9 $\pm$ 4.68	51.8 $\pm$ 5.37
LAMBADA $\uparrow$	6.7 $\pm$ 0.45	5.6 $\pm$ 0.51	4.4 $\pm$ 0.45	6.9 $\pm$ 0.43	5.9 $\pm$ 0.66	4.6 $\pm$ 0.38
PIQA $\uparrow$	55.9 $\pm$ 0.55	55.4 $\pm$ 0.37	55.2 $\pm$ 0.55	55.8 $\pm$ 0.50	55.8 $\pm$ 0.72	54.4 $\pm$ 0.45
ICL composite (macro) $\uparrow$	32.06 $\pm$ 0.25	30.40 $\pm$ 1.40	30.62 $\pm$ 0.66	31.22 $\pm$ 0.69	30.34 $\pm$ 0.92	30.64 $\pm$ 1.10
PPL $\downarrow$	24.0	26.4	32.9	24.0	26.2	32.8

Table E.3 (continued): Anti-induction and Balanced results.

Because any > 0 shot setting exercises in-context learning, I also study 1-shot sensitivity for the same tasks and benchmarks in Appendix E.4.2.

E.4.2 In-context learning robustness

Sensitivity to Number of Shots

I assess how the ICL results in Section 7.3.1 (with details in Appendix E.4.1) vary with the number of in-context demonstrations. Concretely, I change the evaluation from the main-text setting (3-shot for standard LM benchmarks and 10-shot for function-probe tasks) to a unified 1-shot setting for both families. Summaries appear in Figure E.2a (standard LM) and Figure E.2b (function probes); per-task scores are in Table E.5.

For the standard LM benchmarks, moving to 1-shot produces negligible changes across all model sizes (0.13B, 0.5B, 1B). In contrast, the function-style probes degrade notably at 0.5B and 1B when reduced to 1-shot, while the 0.13B model shows only a small drop. This scale-dependent sensitivity aligns with prior observations that larger models more reliably use the demonstration label $\rightarrow$ token mapping (and thus benefit from more shots), whereas smaller models often gain primarily from format/structure and topical priming (Wei et al., 2023; Min et al., 2022; Zhao et al., 2021). Consistently, my label-permutation stress test shows minimal impact at 0.13B but clear degradation at 0.5B/1B, indicating that bigger models lean more on the (now corrupted) mapping signal.

Across shot conditions, the BASELINE vs. BI-INDUCT ordering remains stable: reducing shots changes the absolute level but not the ranking among curricula.

Function-Probe Task Stress Tests

Because stress tests were explicitly considered during the development of the Todd et al. (2024) function-probe suite, I evaluate two that directly target ICL robustness: (i) *label permutation* within the in-context demonstration shots, and (ii) *decision-rule sensitivity*, contrasting the commonly reported HITS@3 with my pri-

	Baseline			Induction		
	0.13B	0.5B	1B	0.13B	0.5B	1B
MMLU	26.7 $\pm$ 0.1	27.5 $\pm$ 0.4	27.7 $\pm$ 0.0	25.9 $\pm$ 0.0	27.4 $\pm$ 0.0	26.6 $\pm$ 0.0
Winogrande	50.4 $\pm$ 0.6	50.8 $\pm$ 1.4	49.8 $\pm$ 1.1	47.0 $\pm$ 0.9	50.5 $\pm$ 1.1	51.1 $\pm$ 0.8
CommonSenseQA	20.8 $\pm$ 1.2	20.0 $\pm$ 1.1	21.2 $\pm$ 0.9	20.3 $\pm$ 0.3	20.5 $\pm$ 0.9	20.5 $\pm$ 0.8
PIQA	56.6 $\pm$ 0.3	58.5 $\pm$ 1.2	59.2 $\pm$ 0.5	55.0 $\pm$ 0.2	58.6 $\pm$ 0.1	58.4 $\pm$ 0.3
HellaSwag	26.5 $\pm$ 0.1	27.1 $\pm$ 0.4	27.8 $\pm$ 0.1	26.3 $\pm$ 0.1	26.5 $\pm$ 0.1	27.2 $\pm$ 0.1
TriviaQA-Wiki	0.1 $\pm$ 0.0	0.2 $\pm$ 0.0	0.4 $\pm$ 0.0	0.1 $\pm$ 0.0	0.1 $\pm$ 0.0	0.3 $\pm$ 0.0
BBH (CoT)	0.1 $\pm$ 0.0	1.5 $\pm$ 0.2	2.8 $\pm$ 0.0	0.3 $\pm$ 0.0	1.6 $\pm$ 0.1	1.2 $\pm$ 0.1
OpenBookQA	14.3 $\pm$ 0.6	14.0 $\pm$ 1.6	15.9 $\pm$ 1.1	13.9 $\pm$ 0.8	15.3 $\pm$ 0.1	15.3 $\pm$ 0.4
ARC-Challenge	18.6 $\pm$ 0.6	18.4 $\pm$ 1.1	18.2 $\pm$ 0.6	18.3 $\pm$ 0.3	18.5 $\pm$ 0.7	18.1 $\pm$ 0.4
GPQA	25.2 $\pm$ 2.2	26.1 $\pm$ 2.1	25.2 $\pm$ 2.0	23.7 $\pm$ 2.3	25.2 $\pm$ 1.7	24.3 $\pm$ 1.5
GSM-8K	1.5 $\pm$ 0.4	1.5 $\pm$ 0.3	1.7 $\pm$ 0.2	1.1 $\pm$ 0.3	1.5 $\pm$ 0.3	1.5 $\pm$ 0.2
MathQA	20.5 $\pm$ 0.4	20.9 $\pm$ 0.7	20.5 $\pm$ 0.7	19.9 $\pm$ 0.6	20.3 $\pm$ 0.2	20.7 $\pm$ 0.6
BoolQ	48.8 $\pm$ 0.5	53.4 $\pm$ 0.9	54.7 $\pm$ 0.2	49.1 $\pm$ 0.7	60.5 $\pm$ 0.3	57.0 $\pm$ 0.9
LAMBADA (OpenAI)	8.2 $\pm$ 0.2	11.0 $\pm$ 0.4	13.2 $\pm$ 0.1	5.5 $\pm$ 0.1	8.6 $\pm$ 0.1	12.2 $\pm$ 0.2
ICL composite (macro) $\uparrow$	22.7 $\pm$ 0.5	23.6 $\pm$ 0.8	24.2 $\pm$ 0.5	21.9 $\pm$ 0.5	23.9 $\pm$ 0.4	23.9 $\pm$ 0.5

Table E.4: Results across model scales (0.13B, 0.5B, 1B) on THE PILE at iso-FLOPs. Copy snippets use span $L=20$. Evaluation is few-shot: 3-shot for standard LM benchmarks, and 10-shot for function-style probes. I report per-task accuracy (or EM where standard), averaged over three seeds, and the ICL composite (macro-average across tasks). Higher is better.

	Anti-induction			Balanced		
	0.13B	0.5B	1B	0.13B	0.5B	1B
MMLU	27.1 ± 0.0	26.8 ± 0.0	27.2 ± 0.1	26.2 ± 0.1	27.1 ± 0.2	27.1 ± 0.0
Winogrande	51.2 ± 0.9	50.1 ± 1.0	49.8 ± 1.3	50.9 ± 1.6	50.9 ± 0.4	51.0 ± 1.2
CommonSenseQA	20.8 ± 0.3	20.0 ± 1.2	20.0 ± 0.4	20.6 ± 0.7	20.5 ± 0.3	20.4 ± 0.4
PIQA	56.1 ± 0.2	56.9 ± 0.3	58.9 ± 0.6	55.5 ± 0.7	57.2 ± 0.6	58.4 ± 0.3
HellaSwag	26.4 ± 0.1	26.8 ± 0.1	27.3 ± 0.1	26.2 ± 0.1	26.5 ± 0.1	27.3 ± 0.1
TriviaQA-Wiki	0.1 ± 0.0	0.1 ± 0.0	0.3 ± 0.0	0.1 ± 0.0	0.1 ± 0.0	0.3 ± 0.0
BBH (CoT)	0.8 ± 0.1	0.1 ± 0.0	0.6 ± 0.0	0.1 ± 0.0	2.2 ± 0.0	4.2 ± 0.0
OpenBookQA	14.7 ± 0.7	13.9 ± 1.3	15.6 ± 0.4	13.1 ± 0.1	13.6 ± 1.0	16.5 ± 0.5
ARC-Challenge	17.6 ± 0.4	17.8 ± 0.4	19.0 ± 0.4	17.1 ± 0.8	17.9 ± 0.4	17.9 ± 0.3
GPQA	23.9 ± 2.0	24.9 ± 1.4	24.1 ± 1.1	24.1 ± 2.2	22.8 ± 0.8	24.6 ± 0.8
GSM-8K	1.4 ± 0.2	1.2 ± 0.1	1.6 ± 0.1	1.1 ± 0.4	1.7 ± 0.2	1.4 ± 0.2
MathQA	20.2 ± 0.3	20.4 ± 0.4	21.1 ± 0.2	21.0 ± 0.7	20.8 ± 0.4	21.0 ± 0.1
BoolQ	49.1 ± 0.8	52.2 ± 2.2	52.7 ± 0.3	51.4 ± 0.5	57.1 ± 1.0	58.1 ± 0.4
LAMBADA (OpenAI)	5.2 ± 0.2	8.4 ± 0.3	12.2 ± 0.2	6.0 ± 0.2	8.2 ± 0.2	12.0 ± 0.3
ICL composite (macro) ↑	22.5 ± 0.4	22.8 ± 0.6	23.6 ± 0.4	22.4 ± 0.6	23.3 ± 0.4	24.3 ± 0.3

Table E.4 (continued): Anti-induction and Balanced standard-benchmark results.

	Baseline				Induction			
	0.13B	0.5B	1B	0.13B	0.5B	1B		
alphabetically_first_3	4.9 $\pm$ 0.6	9.4 $\pm$ 0.6	19.7 $\pm$ 0.9	4.6 $\pm$ 0.6	7.3 $\pm$ 0.3	15.1 $\pm$ 0.9		
alphabetically_first_5	4.0 $\pm$ 0.6	6.7 $\pm$ 0.8	12.4 $\pm$ 1.4	4.2 $\pm$ 0.8	6.5 $\pm$ 1.0	10.9 $\pm$ 0.9		
alphabetically_last_3	3.4 $\pm$ 0.5	10.2 $\pm$ 0.9	20.8 $\pm$ 0.6	2.4 $\pm$ 0.5	7.7 $\pm$ 0.2	15.8 $\pm$ 1.0		
alphabetically_last_5	2.5 $\pm$ 0.7	6.0 $\pm$ 1.3	9.9 $\pm$ 0.8	1.6 $\pm$ 0.3	5.5 $\pm$ 0.4	8.5 $\pm$ 0.3		
capitalize	8.0 $\pm$ 1.0	20.7 $\pm$ 1.4	54.8 $\pm$ 2.1	3.3 $\pm$ 0.3	13.2 $\pm$ 1.3	33.4 $\pm$ 1.3		
capitalize_first_letter	10.1 $\pm$ 1.2	12.5 $\pm$ 1.8	28.6 $\pm$ 1.1	5.3 $\pm$ 1.0	13.4 $\pm$ 1.7	13.7 $\pm$ 1.2		
capitalize_last_letter	4.8 $\pm$ 1.3	9.6 $\pm$ 1.0	8.3 $\pm$ 1.2	8.6 $\pm$ 1.1	7.5 $\pm$ 1.8	8.6 $\pm$ 0.9		
choose_first_of_3	10.3 $\pm$ 2.3	25.5 $\pm$ 1.8	69.4 $\pm$ 1.8	6.1 $\pm$ 0.9	19.0 $\pm$ 1.7	52.4 $\pm$ 2.0		
choose_first_of_5	8.7 $\pm$ 1.3	19.7 $\pm$ 1.9	55.5 $\pm$ 1.3	4.9 $\pm$ 0.7	15.3 $\pm$ 0.9	42.6 $\pm$ 3.0		
choose_last_of_3	2.0 $\pm$ 0.4	3.2 $\pm$ 0.2	4.4 $\pm$ 1.0	1.4 $\pm$ 0.4	2.8 $\pm$ 0.3	5.4 $\pm$ 0.9		
choose_last_of_5	1.5 $\pm$ 0.3	2.9 $\pm$ 0.6	3.9 $\pm$ 1.1	1.7 $\pm$ 0.5	2.4 $\pm$ 0.4	4.6 $\pm$ 0.6		
choose_middle_of_3	1.7 $\pm$ 0.6	3.3 $\pm$ 0.5	4.2 $\pm$ 0.5	2.1 $\pm$ 0.8	2.2 $\pm$ 0.3	6.0 $\pm$ 1.0		
choose_middle_of_5	1.6 $\pm$ 0.3	3.0 $\pm$ 0.6	2.7 $\pm$ 0.4	1.6 $\pm$ 0.2	2.1 $\pm$ 0.3	3.1 $\pm$ 0.6		
lowercase_first_letter	5.5 $\pm$ 0.8	8.4 $\pm$ 0.9	28.5 $\pm$ 2.2	4.1 $\pm$ 0.9	6.1 $\pm$ 1.2	20.1 $\pm$ 1.6		
lowercase_last_letter	11.1 $\pm$ 0.8	7.7 $\pm$ 0.7	10.5 $\pm$ 1.1	3.6 $\pm$ 0.7	8.0 $\pm$ 0.7	9.6 $\pm$ 1.0		
next_capital_letter	4.9 $\pm$ 1.1	4.2 $\pm$ 0.9	2.4 $\pm$ 0.8	3.9 $\pm$ 0.3	4.2 $\pm$ 1.0	3.6 $\pm$ 0.7		
next_item	3.4 $\pm$ 2.2	8.6 $\pm$ 2.5	16.8 $\pm$ 2.1	2.9 $\pm$ 1.9	5.7 $\pm$ 1.3	11.8 $\pm$ 1.2		
prev_item	2.5 $\pm$ 0.8	7.7 $\pm$ 1.6	15.1 $\pm$ 2.3	2.0 $\pm$ 0.8	5.9 $\pm$ 0.7	10.3 $\pm$ 1.0		
word_length	9.9 $\pm$ 1.0	13.3 $\pm$ 0.9	12.8 $\pm$ 1.1	13.4 $\pm$ 0.8	14.0 $\pm$ 1.0	13.3 $\pm$ 0.8		
ICL composite (macro) $\uparrow$	5.3 $\pm$ 0.9	9.6 $\pm$ 1.1	20.0 $\pm$ 1.3	4.1 $\pm$ 0.7	7.8 $\pm$ 0.9	15.2 $\pm$ 1.1		

Table E.4 (continued): function-style probe results.

	Anti-induction			Balanced		
	0.13B	0.5B	1B	0.13B	0.5B	1B
alphabetically_first-3	3.3 ± 0.5	8.5 ± 0.9	14.4 ± 1.5	4.8 ± 0.6	11.5 ± 0.6	15.3 ± 1.0
alphabetically_first-5	2.8 ± 0.7	7.4 ± 0.6	8.4 ± 1.1	3.9 ± 1.0	8.2 ± 0.8	10.2 ± 0.7
alphabetically_last-3	1.9 ± 0.6	8.0 ± 0.9	13.3 ± 0.8	3.9 ± 0.5	10.9 ± 0.5	15.3 ± 1.3
alphabetically_last-5	1.8 ± 0.4	6.1 ± 0.8	7.8 ± 0.5	2.5 ± 0.8	6.7 ± 0.6	9.3 ± 0.4
capitalize	3.7 ± 0.1	13.5 ± 1.3	39.2 ± 2.7	6.0 ± 1.6	14.7 ± 1.3	33.6 ± 2.1
capitalize_first_letter	5.2 ± 0.3	11.2 ± 1.2	17.4 ± 0.8	8.9 ± 0.7	10.0 ± 1.1	20.4 ± 1.2
capitalize_last_letter	9.1 ± 1.4	9.3 ± 1.2	7.3 ± 1.5	5.5 ± 0.7	5.8 ± 1.4	6.7 ± 0.7
choose_first_of-3	4.1 ± 0.7	19.1 ± 1.8	46.0 ± 2.0	11.3 ± 0.7	35.2 ± 1.7	54.1 ± 1.7
choose_first_of-5	3.6 ± 0.7	16.2 ± 1.5	32.7 ± 1.6	7.9 ± 1.0	28.2 ± 2.4	42.1 ± 1.4
choose_last_of-3	1.4 ± 0.5	3.0 ± 0.5	5.0 ± 0.5	1.7 ± 0.3	3.0 ± 0.5	5.2 ± 0.3
choose_last_of-5	1.1 ± 0.4	2.9 ± 0.5	5.6 ± 0.5	1.5 ± 0.3	2.6 ± 0.4	5.1 ± 0.4
choose_middle_of-3	1.3 ± 0.5	3.1 ± 0.2	5.0 ± 0.3	1.7 ± 0.7	3.4 ± 0.7	4.9 ± 0.7
choose_middle_of-5	1.7 ± 0.4	2.9 ± 0.7	3.9 ± 0.7	1.7 ± 0.4	2.3 ± 0.6	4.7 ± 0.6
lowercase_first_letter	4.7 ± 0.7	6.5 ± 0.6	20.6 ± 1.1	2.2 ± 0.6	8.8 ± 0.7	13.3 ± 0.8
lowercase_last_letter	7.4 ± 1.2	8.9 ± 1.0	13.3 ± 0.5	7.8 ± 1.1	10.9 ± 1.5	8.9 ± 1.2
next_capital_letter	4.7 ± 0.8	4.1 ± 0.7	3.1 ± 0.9	4.8 ± 0.9	3.5 ± 0.9	4.2 ± 1.1
next_item	1.3 ± 0.9	7.8 ± 1.1	11.8 ± 2.5	6.3 ± 0.8	9.0 ± 1.6	9.4 ± 1.7
prev_item	1.4 ± 0.5	6.6 ± 2.5	11.4 ± 2.5	5.7 ± 1.3	8.4 ± 1.2	7.7 ± 2.2
word_length	12.0 ± 1.3	14.3 ± 0.6	13.6 ± 1.6	10.8 ± 1.1	14.7 ± 1.8	12.6 ± 0.7
ICL composite (macro) ↑	3.8 ± 0.7	8.4 ± 1.0	14.7 ± 1.2	5.2 ± 0.8	10.4 ± 1.1	14.9 ± 1.1

Table E.4 (continued): Anti-induction and Balanced function-style probe results.

	Baseline			Induction		
	0.13B	0.5B	1B	0.13B	0.5B	1B
MMLU	25.8 $\pm$ 0.0	25.2 $\pm$ 0.3	26.8 $\pm$ 0.0	25.8 $\pm$ 0.0	24.4 $\pm$ 0.3	26.4 $\pm$ 0.0
Winogrande	49.8 $\pm$ 0.8	50.3 $\pm$ 0.5	49.6 $\pm$ 0.4	48.9 $\pm$ 0.3	50.2 $\pm$ 0.3	50.6 $\pm$ 0.4
CommonSenseQA	20.9 $\pm$ 0.9	20.6 $\pm$ 1.0	20.5 $\pm$ 0.5	20.8 $\pm$ 0.8	20.6 $\pm$ 0.9	20.6 $\pm$ 0.6
PIQA	56.8 $\pm$ 0.4	58.7 $\pm$ 0.8	59.9 $\pm$ 0.4	55.4 $\pm$ 0.2	58.8 $\pm$ 0.2	58.3 $\pm$ 0.3
HellaSwag	26.4 $\pm$ 0.1	27.0 $\pm$ 0.1	27.8 $\pm$ 0.1	26.2 $\pm$ 0.1	26.7 $\pm$ 0.0	27.3 $\pm$ 0.1
TriviaQA-Wiki	0.1 $\pm$ 0.0	0.1 $\pm$ 0.0	0.3 $\pm$ 0.0	0.1 $\pm$ 0.0	0.1 $\pm$ 0.0	0.1 $\pm$ 0.0
BBH (CoT)	0.0 $\pm$ 0.0	0.6 $\pm$ 0.0	3.4 $\pm$ 0.0	0.1 $\pm$ 0.02	2.0 $\pm$ 0.0	4.3 $\pm$ 0.0
OpenBookQA	14.3 $\pm$ 0.5	15.6 $\pm$ 0.7	15.8 $\pm$ 0.5	14.3 $\pm$ 0.2	14.5 $\pm$ 0.8	15.5 $\pm$ 0.5
ARC-Challenge	18.4 $\pm$ 0.2	19.0 $\pm$ 0.4	18.1 $\pm$ 0.1	18.7 $\pm$ 0.4	18.2 $\pm$ 0.1	18.3 $\pm$ 0.2
GPQA	24.8 $\pm$ 0.6	25.0 $\pm$ 1.0	26.2 $\pm$ 2.5	25.0 $\pm$ 0.8	24.4 $\pm$ 1.3	25.2 $\pm$ 0.6
GSM-8K	1.3 $\pm$ 0.1	1.9 $\pm$ 0.3	1.3 $\pm$ 0.1	1.5 $\pm$ 0.6	2.2 $\pm$ 0.3	1.5 $\pm$ 0.2
MathQA	20.7 $\pm$ 0.3	20.4 $\pm$ 0.2	20.7 $\pm$ 0.3	20.3 $\pm$ 0.5	20.2 $\pm$ 0.3	21.1 $\pm$ 0.3
BoolQ	53.2 $\pm$ 0.8	53.4 $\pm$ 0.8	54.2 $\pm$ 0.3	53.2 $\pm$ 0.8	57.6 $\pm$ 1.1	56.0 $\pm$ 0.2
LAMBADA	9.5 $\pm$ 0.1	11.9 $\pm$ 0.1	15.1 $\pm$ 0.1	6.5 $\pm$ 0.1	9.9 $\pm$ 0.2	13.4 $\pm$ 0.2
ICL composite (macro)	23.0 $\pm$ 0.3	23.6 $\pm$ 0.4	24.3 $\pm$ 0.4	22.6 $\pm$ 0.3	23.6 $\pm$ 0.4	24.2 $\pm$ 0.3

Table E.5: Results across model scales (0.13B, 0.5B, 1B) on THE PILE at iso-FLOPs. Copy snippets use span $L=20$. Evaluation is few-shot: 1-shot for both standard LM benchmarks, and function-style probes. I report per-task accuracy (or EM where standard), averaged over three seeds, and the ICL composite (macro-average across tasks). Higher is better.

	Anti-induction			Balanced		
	0.13B	0.5B	1B	0.13B	0.5B	1B
MMU	25.8 ± 0.0	24.7 ± 0.3	26.8 ± 0.0	25.9 ± 0.0	24.3 ± 0.2	27.3 ± 0.0
Winogrande	50.9 ± 1.2	50.4 ± 0.8	49.3 ± 0.9	50.8 ± 0.3	52.0 ± 0.4	51.0 ± 0.0
CommonSenseQA	20.9 ± 0.8	20.6 ± 1.0	20.7 ± 0.1	21.0 ± 0.8	20.7 ± 0.9	20.6 ± 0.6
PIQA	55.9 ± 0.1	57.4 ± 0.5	59.0 ± 1.1	55.4 ± 0.9	57.3 ± 0.3	58.1 ± 0.2
HellaSwag	26.3 ± 0.1	26.7 ± 0.2	27.3 ± 0.1	26.2 ± 0.1	26.6 ± 0.1	27.2 ± 0.2
TriviaQA-Wiki	0.1 ± 0.0	0.1 ± 0.0	0.1 ± 0.0	0.1 ± 0.0	0.1 ± 0.0	0.2 ± 0.0
BBH (CoT)	0.4 ± 0.00	0.1 ± 0.0	0.6 ± 0.1	0.4 ± 0.00	1.8 ± 1.1	2.8 ± 0.0
OpenBookQA	13.0 ± 0.0	14.7 ± 0.2	16.7 ± 0.7	13.3 ± 0.1	13.9 ± 0.1	15.6 ± 1.2
ARC-Challenge	17.7 ± 0.3	17.8 ± 0.2	18.4 ± 0.5	17.5 ± 0.4	17.8 ± 0.4	17.7 ± 0.7
GPQA	25.1 ± 1.1	25.9 ± 1.2	25.7 ± 1.2	25.5 ± 0.3	24.2 ± 2.1	26.0 ± 2.0
GSM-8K	1.3 ± 0.5	1.8 ± 0.2	1.6 ± 0.3	1.1 ± 0.1	2.0 ± 0.3	1.5 ± 0.2
MathQA	20.1 ± 0.1	20.3 ± 0.1	20.8 ± 0.7	20.6 ± 0.4	20.2 ± 0.3	21.0 ± 1.0
BoolQ	50.8 ± 0.8	53.0 ± 0.7	54.1 ± 0.1	53.1 ± 0.8	53.1 ± 0.7	55.2 ± 0.4
LAMBADA	5.9 ± 0.2	9.6 ± 0.3	13.5 ± 0.2	7.0 ± 0.1	9.2 ± 0.1	13.8 ± 0.5
ICL composite (macro)	22.4 ± 0.4	23.1 ± 0.4	23.9 ± 0.4	22.7 ± 0.3	23.1 ± 0.5	24.1 ± 0.5

Table E.5 (continued): Anti-induction and Balanced standard-benchmark results.

	Baseline			Induction		
	0.13B	0.5B	1B	0.13B	0.5B	1B
alphabetically_first_3	2.6 $\pm$ 0.6	3.0 $\pm$ 0.7	13.2 $\pm$ 1.3	3.3 $\pm$ 0.7	4.8 $\pm$ 0.6	8.9 $\pm$ 0.8
alphabetically_first_5	2.7 $\pm$ 0.4	3.3 $\pm$ 0.5	8.9 $\pm$ 0.5	3.3 $\pm$ 0.1	4.3 $\pm$ 0.2	7.5 $\pm$ 0.4
alphabetically_last_3	2.1 $\pm$ 0.5	2.9 $\pm$ 0.5	13.3 $\pm$ 1.2	3.1 $\pm$ 0.4	4.3 $\pm$ 0.9	9.2 $\pm$ 1.0
alphabetically_last_5	1.6 $\pm$ 0.5	2.8 $\pm$ 0.4	8.5 $\pm$ 0.9	1.9 $\pm$ 0.6	3.2 $\pm$ 0.4	6.0 $\pm$ 0.3
capitalize	2.9 $\pm$ 0.6	1.5 $\pm$ 0.4	4.4 $\pm$ 0.7	1.8 $\pm$ 0.3	5.1 $\pm$ 0.8	2.6 $\pm$ 0.8
capitalize_first_letter	4.1 $\pm$ 1.1	3.0 $\pm$ 0.9	3.4 $\pm$ 0.7	3.5 $\pm$ 1.0	5.6 $\pm$ 1.4	4.0 $\pm$ 1.2
capitalize_last_letter	9.0 $\pm$ 0.7	7.4 $\pm$ 0.4	5.6 $\pm$ 0.9	9.1 $\pm$ 0.8	6.0 $\pm$ 0.7	8.4 $\pm$ 0.4
choose_first_of_3	5.6 $\pm$ 0.5	6.7 $\pm$ 1.0	37.0 $\pm$ 1.2	5.5 $\pm$ 0.8	7.4 $\pm$ 1.3	25.1 $\pm$ 2.1
choose_first_of_5	5.9 $\pm$ 0.4	6.9 $\pm$ 0.8	32.0 $\pm$ 1.2	3.9 $\pm$ 0.8	5.6 $\pm$ 0.6	24.9 $\pm$ 1.9
choose_last_of_3	0.8 $\pm$ 0.3	1.1 $\pm$ 0.3	3.7 $\pm$ 0.6	1.3 $\pm$ 0.4	1.9 $\pm$ 0.3	2.4 $\pm$ 0.5
choose_last_of_5	0.9 $\pm$ 0.3	1.1 $\pm$ 0.4	2.4 $\pm$ 0.4	1.4 $\pm$ 0.3	1.5 $\pm$ 0.3	2.0 $\pm$ 0.4
choose_middle_of_3	0.7 $\pm$ 0.3	1.1 $\pm$ 0.4	3.5 $\pm$ 0.6	0.9 $\pm$ 0.1	1.6 $\pm$ 0.4	2.5 $\pm$ 0.6
choose_middle_of_5	0.9 $\pm$ 0.4	1.4 $\pm$ 0.5	2.3 $\pm$ 0.5	1.2 $\pm$ 0.3	1.4 $\pm$ 0.2	2.2 $\pm$ 0.5
lowercase_first_letter	4.9 $\pm$ 0.7	3.7 $\pm$ 0.7	4.0 $\pm$ 0.3	4.0 $\pm$ 0.7	3.9 $\pm$ 0.6	4.6 $\pm$ 0.8
lowercase_last_letter	10.6 $\pm$ 1.2	9.3 $\pm$ 1.2	9.5 $\pm$ 1.2	6.6 $\pm$ 0.9	7.7 $\pm$ 1.3	10.6 $\pm$ 1.2
next_capital_letter	4.5 $\pm$ 0.3	3.8 $\pm$ 0.6	3.6 $\pm$ 1.0	4.6 $\pm$ 0.5	3.9 $\pm$ 0.4	4.5 $\pm$ 0.4
next_item	2.5 $\pm$ 0.4	2.0 $\pm$ 1.7	9.7 $\pm$ 1.0	3.9 $\pm$ 0.8	5.1 $\pm$ 1.4	5.1 $\pm$ 1.2
prev_item	2.5 $\pm$ 1.2	2.3 $\pm$ 1.7	9.2 $\pm$ 1.6	3.0 $\pm$ 1.1	5.1 $\pm$ 1.6	5.7 $\pm$ 2.0
word_length	13.8 $\pm$ 1.9	13.7 $\pm$ 1.7	12.7 $\pm$ 2.2	12.7 $\pm$ 1.5	13.4 $\pm$ 2.1	13.6 $\pm$ 1.5
ICL composite (macro) $\uparrow$	4.1 $\pm$ 0.6	4.1 $\pm$ 0.8	9.8 $\pm$ 0.9	3.9 $\pm$ 0.6	4.8 $\pm$ 0.8	7.9 $\pm$ 1.0

Table E.5 (continued): function-style probe results.

	Anti-induction			Balanced		
	0.13B	0.5B	1B	0.13B	0.5B	1B
alphabetically_first-3	1.9 ± 0.4	3.5 ± 0.8	5.8 ± 1.0	3.6 ± 0.7	4.4 ± 0.4	8.5 ± 0.8
alphabetically_first-5	2.0 ± 0.3	3.7 ± 0.8	4.4 ± 0.8	2.2 ± 0.4	3.1 ± 0.6	6.8 ± 1.1
alphabetically_last-3	1.9 ± 0.4	2.9 ± 0.3	6.8 ± 0.6	4.5 ± 0.3	3.8 ± 0.5	10.6 ± 0.9
alphabetically_last-5	1.4 ± 0.4	2.3 ± 0.5	4.5 ± 0.4	2.2 ± 0.4	3.0 ± 0.8	6.8 ± 0.6
capitalize	1.8 ± 0.2	2.5 ± 0.5	4.9 ± 0.6	2.7 ± 0.4	3.1 ± 0.5	3.7 ± 1.1
capitalize_first_letter	3.5 ± 1.0	4.7 ± 1.2	4.0 ± 0.9	4.4 ± 1.0	4.9 ± 0.8	3.5 ± 1.1
capitalize_last_letter	9.2 ± 0.8	8.7 ± 0.9	8.4 ± 0.8	8.7 ± 0.7	8.8 ± 0.8	8.8 ± 0.5
choose_first-of-3	1.5 ± 0.4	6.0 ± 1.1	14.6 ± 1.0	8.9 ± 1.0	7.2 ± 0.8	26.1 ± 1.8
choose_first-of-5	1.1 ± 0.3	6.1 ± 0.8	12.5 ± 1.9	5.3 ± 0.3	4.8 ± 0.6	23.8 ± 1.4
choose_last-of-3	1.2 ± 0.4	1.3 ± 0.7	2.0 ± 0.4	0.7 ± 0.2	1.9 ± 0.7	3.7 ± 0.3
choose_last-of-5	0.9 ± 0.3	1.3 ± 0.4	1.9 ± 0.6	0.6 ± 0.2	1.3 ± 0.4	3.1 ± 0.6
choose_middle-of-3	0.7 ± 0.2	1.1 ± 0.1	1.9 ± 0.5	0.6 ± 0.1	1.1 ± 0.2	3.6 ± 0.4
choose_middle-of-5	1.2 ± 0.3	1.3 ± 0.5	1.6 ± 0.4	0.8 ± 0.4	1.5 ± 0.7	3.0 ± 0.6
lowercase_first_letter	2.6 ± 0.5	4.7 ± 0.7	4.7 ± 0.8	2.3 ± 0.5	4.7 ± 0.7	4.4 ± 0.8
lowercase_last_letter	7.6 ± 0.5	10.6 ± 1.2	10.6 ± 1.2	8.7 ± 1.3	10.6 ± 1.2	10.3 ± 1.2
next_capital_letter	4.4 ± 0.4	4.7 ± 0.1	4.1 ± 0.5	4.1 ± 0.4	4.5 ± 0.4	4.6 ± 0.5
next_item	1.5 ± 1.1	2.8 ± 1.1	3.5 ± 0.8	4.4 ± 2.1	4.2 ± 1.1	3.4 ± 1.5
prev_item	1.8 ± 0.7	3.3 ± 1.9	3.4 ± 1.9	4.6 ± 0.7	3.0 ± 1.5	3.8 ± 1.3
word_length	12.5 ± 1.6	13.9 ± 1.7	13.9 ± 1.4	10.6 ± 1.0	13.9 ± 1.7	13.9 ± 1.9
ICL composite (macro) ↑	3.1 ± 0.5	4.5 ± 0.8	6.0 ± 0.9	4.2 ± 0.6	4.7 ± 0.7	8.0 ± 1.0

Table E.5 (continued): Anti-induction and Balanced function-style probe results.

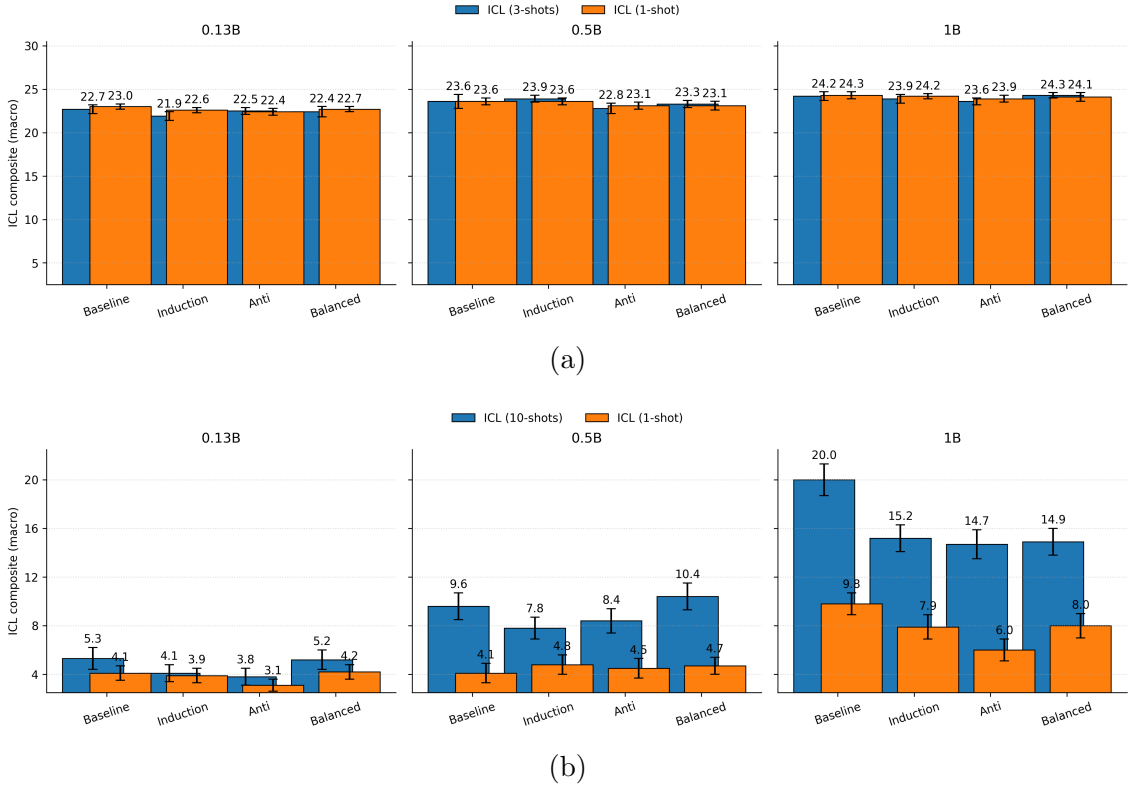


Figure E.2: Sensitivity of ICL composite (macro) to the number of shots across two evaluation families: (a) standard LM benchmarks (3-shot vs. 1-shot); (b) Function-probe suite of Todd et al. (2024) (10-shot vs. 1-shot). Each panel groups models by size (0.13B, 0.5B, 1B), colours the bars by regime (Baseline, Induction, Anti, Balanced), and shows ± 1 standard deviation error bars.

mary metric, HITS@1. These tests probe robustness to spurious label-token mappings and to the choice of evaluation rule, respectively.

Label-Permutation Stress Test: I stress-test in-context usage on the Todd et al. (2024) probes by randomly permuting the targets within the 10 demonstration shots (inputs unchanged) and evaluating on the true task distribution. If a model relies on the demonstrations, accuracy should drop; if it leans on parametric priors, it should be less affected. As shown in Figure E.3 and Table E.6, the relative ordering between Baseline and Bi-Induct variants is largely preserved. At **0.13B**, permutation produces no degradation for either, suggesting heavier reliance on parametric knowledge. At **0.5B** and **1B**, all curricula degrade, indicating increased sensitivity to the in-context mapping. Overall, Bi-Induct mirrors Baseline at each scale: robust at 0.13B and increasingly demonstration-sensitive as scale grows.

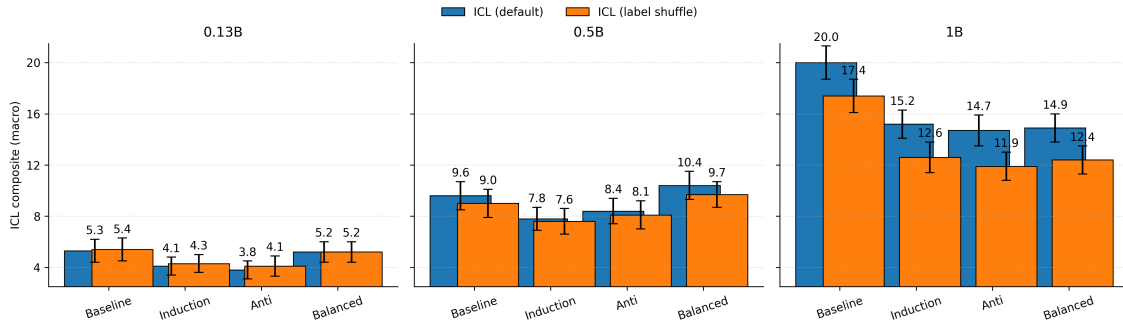


Figure E.3: Function-probe suite of Todd et al. (2024) - ICL Composite (macro). Three panels (0.13B, 0.5B, 1B). For each model, bars compare *ICL (default - no label shuffle)* vs *ICL (label shuffle)* across four regimes (Baseline, Induction, Anti, Balanced). Error bars show ± 1 standard deviation

Why permutation hurts 0.5B-1B but not 0.13B? At 0.13B, the robustness to label permutation suggests it benefits from demonstrations via format/topical priming and answer-frequency priors, but does not reliably exploit the label→token mapping. In contrast, 0.5B-1B models more strongly use the in-context mapping; permuting labels therefore contradicts a cue they have learned to trust, producing clear drops. This is consistent with reports that (i) labels in demos can be non-essential for smaller/less capable settings-format and priors often dominate (Min et al., 2022; Zhao et al., 2021), and (ii) the ability to override priors and follow contradictory, flipped labels emerges with scale (Wei et al., 2023).

	Baseline			Induction		
	0.13B	0.5B	1B	0.13B	0.5B	1B
alphabetically_first_3	4.4 $\pm$ 0.4	9.2 $\pm$ 0.7	16.6 $\pm$ 0.6	4.4 $\pm$ 0.3	7.0 $\pm$ 0.7	12.3 $\pm$ 1.4
alphabetically_first_5	4.1 $\pm$ 0.7	7.1 $\pm$ 1.1	11.1 $\pm$ 0.9	4.2 $\pm$ 0.6	5.9 $\pm$ 0.4	9.4 $\pm$ 0.5
alphabetically_last_3	3.2 $\pm$ 0.6	9.7 $\pm$ 0.8	16.1 $\pm$ 1.6	2.6 $\pm$ 0.6	7.3 $\pm$ 0.9	12.3 $\pm$ 0.6
alphabetically_last_5	2.6 $\pm$ 0.3	5.8 $\pm$ 0.9	9.5 $\pm$ 0.2	1.6 $\pm$ 0.3	5.3 $\pm$ 0.3	7.8 $\pm$ 0.7
capitalize	8.7 $\pm$ 0.8	18.2 $\pm$ 1.9	49.3 $\pm$ 2.9	3.6 $\pm$ 1.0	13.3 $\pm$ 1.8	29.9 $\pm$ 0.8
capitalize_first_letter	10.4 $\pm$ 2.1	13.2 $\pm$ 1.4	29.0 $\pm$ 1.4	6.4 $\pm$ 0.4	13.3 $\pm$ 0.8	15.2 $\pm$ 1.4
capitalize_last_letter	4.7 $\pm$ 1.0	8.2 $\pm$ 1.0	7.7 $\pm$ 0.7	7.9 $\pm$ 1.2	6.7 $\pm$ 0.8	7.4 $\pm$ 1.3
choose_first_of_3	11.2 $\pm$ 1.2	21.9 $\pm$ 2.1	54.9 $\pm$ 2.6	6.9 $\pm$ 1.2	19.1 $\pm$ 2.5	35.0 $\pm$ 2.5
choose_first_of_5	9.3 $\pm$ 1.5	17.3 $\pm$ 1.6	42.0 $\pm$ 2.3	5.6 $\pm$ 0.9	15.3 $\pm$ 1.8	28.9 $\pm$ 2.1
choose_last_of_3	1.6 $\pm$ 0.4	2.7 $\pm$ 0.6	4.3 $\pm$ 0.9	2.1 $\pm$ 0.6	2.6 $\pm$ 0.4	4.9 $\pm$ 0.5
choose_last_of_5	1.7 $\pm$ 0.4	2.2 $\pm$ 0.6	3.9 $\pm$ 0.8	1.9 $\pm$ 0.4	2.1 $\pm$ 0.3	4.5 $\pm$ 0.6
choose_middle_of_3	1.9 $\pm$ 0.3	3.5 $\pm$ 0.5	4.5 $\pm$ 0.9	2.3 $\pm$ 0.6	2.8 $\pm$ 0.7	5.6 $\pm$ 1.1
choose_middle_of_5	1.8 $\pm$ 0.7	3.6 $\pm$ 0.8	2.9 $\pm$ 0.5	1.9 $\pm$ 0.3	2.2 $\pm$ 0.6	3.3 $\pm$ 0.4
lowercase_first_letter	6.7 $\pm$ 0.8	9.3 $\pm$ 0.6	27.8 $\pm$ 1.6	4.9 $\pm$ 0.7	6.5 $\pm$ 0.4	19.0 $\pm$ 1.1
lowercase_last_letter	10.1 $\pm$ 1.0	7.5 $\pm$ 1.2	8.8 $\pm$ 1.9	3.5 $\pm$ 1.2	8.0 $\pm$ 1.6	8.2 $\pm$ 1.2
next_capital_letter	4.3 $\pm$ 0.9	4.3 $\pm$ 0.5	2.8 $\pm$ 0.4	4.2 $\pm$ 0.6	3.5 $\pm$ 0.6	3.6 $\pm$ 0.6
next_item	3.3 $\pm$ 1.0	7.3 $\pm$ 1.3	15.3 $\pm$ 1.8	2.7 $\pm$ 0.8	4.7 $\pm$ 0.8	10.8 $\pm$ 2.2
prev_item	3.2 $\pm$ 1.7	6.5 $\pm$ 2.3	13.5 $\pm$ 1.6	2.5 $\pm$ 1.4	5.6 $\pm$ 1.4	8.6 $\pm$ 1.8
word_length	9.7 $\pm$ 0.8	12.6 $\pm$ 1.3	11.4 $\pm$ 1.8	12.8 $\pm$ 1.0	12.9 $\pm$ 1.4	12.4 $\pm$ 1.1
ICL composite (macro) $\uparrow$	5.4 $\pm$ 0.9	9.0 $\pm$ 1.1	17.4 $\pm$ 1.3	4.3 $\pm$ 0.7	7.6 $\pm$ 1.0	12.6 $\pm$ 1.2

Table E.6: Function-probe suite of Todd et al. (2024) under label-permutation stress (Todd et al. (2024) suite): HITS@1 accuracy on 10-shot prompts with demonstration labels randomly permuted; reported as mean $\pm$ std across three seeds for 0.13B, 0.5B, and 1B, comparing Baseline, Induction, Anti, and Balanced curricula.

	Anti-induction			Balanced		
	0.13B	0.5B	1B	0.13B	0.5B	1B
alphabetically_first_3	3.6 ± 0.5	8.2 ± 0.8	11.0 ± 1.3	4.2 ± 0.7	10.7 ± 1.4	12.5 ± 0.8
alphabetically_first_5	2.6 ± 0.9	7.0 ± 0.9	7.0 ± 0.4	3.5 ± 0.4	7.8 ± 1.0	9.0 ± 0.5
alphabetically_last_3	2.7 ± 1.0	7.4 ± 1.3	10.4 ± 0.9	4.8 ± 0.6	10.7 ± 1.4	13.0 ± 0.6
alphabetically_last_5	1.6 ± 0.5	5.5 ± 0.6	6.4 ± 0.8	2.0 ± 0.2	6.7 ± 0.5	8.5 ± 0.9
capitalize	4.3 ± 0.7	13.5 ± 1.6	34.3 ± 1.4	6.8 ± 0.6	14.5 ± 0.9	31.0 ± 1.9
capitalize_first_letter	6.4 ± 0.9	12.2 ± 1.9	17.5 ± 1.3	9.2 ± 1.5	10.8 ± 2.2	21.2 ± 0.7
capitalize_last_letter	8.9 ± 1.7	8.6 ± 0.9	5.9 ± 1.4	5.2 ± 0.7	5.3 ± 0.9	6.5 ± 1.4
choose_first_of_3	5.2 ± 0.4	19.0 ± 2.0	28.9 ± 0.8	13.4 ± 1.0	32.4 ± 1.1	34.5 ± 1.6
choose_first_of_5	3.6 ± 0.4	16.8 ± 1.9	20.3 ± 1.3	8.5 ± 1.7	23.7 ± 1.9	26.4 ± 1.9
choose_last_of_3	1.3 ± 0.4	2.8 ± 0.3	4.9 ± 1.2	1.3 ± 0.3	3.3 ± 0.7	5.4 ± 0.6
choose_last_of_5	1.1 ± 0.4	2.3 ± 0.6	5.5 ± 0.7	1.2 ± 0.5	2.3 ± 0.8	4.7 ± 0.3
choose_middle_of_3	1.4 ± 0.5	3.0 ± 0.4	4.6 ± 0.3	1.5 ± 0.2	3.1 ± 0.3	5.2 ± 0.8
choose_middle_of_5	1.5 ± 0.6	2.8 ± 0.4	3.9 ± 0.5	1.7 ± 0.2	2.4 ± 0.4	4.6 ± 0.7
lowercase_first_letter	5.4 ± 1.1	7.5 ± 1.1	18.4 ± 1.9	2.5 ± 0.6	10.4 ± 0.9	13.9 ± 1.4
lowercase_last_letter	7.1 ± 0.9	8.4 ± 1.7	12.8 ± 1.8	7.1 ± 0.7	9.5 ± 1.3	7.8 ± 1.5
next_capital_letter	4.7 ± 0.6	4.2 ± 0.7	3.0 ± 0.6	4.3 ± 0.5	3.3 ± 0.6	3.0 ± 0.1
next_item	1.8 ± 1.0	4.9 ± 1.2	9.7 ± 2.7	5.3 ± 1.5	6.6 ± 1.4	7.2 ± 1.0
prev_item	2.3 ± 1.5	4.9 ± 1.1	9.2 ± 1.1	4.8 ± 1.7	6.5 ± 1.4	7.8 ± 1.0
word_length	12.1 ± 1.1	14.5 ± 1.6	13.2 ± 0.2	11.1 ± 0.9	14.3 ± 0.8	12.6 ± 1.0
ICL composite (macro) ↑	4.1 ± 0.8	8.1 ± 1.1	11.9 ± 1.1	5.2 ± 0.8	9.7 ± 1.0	12.4 ± 1.0

Table E.6 (continued): Anti-induction and Balanced results.

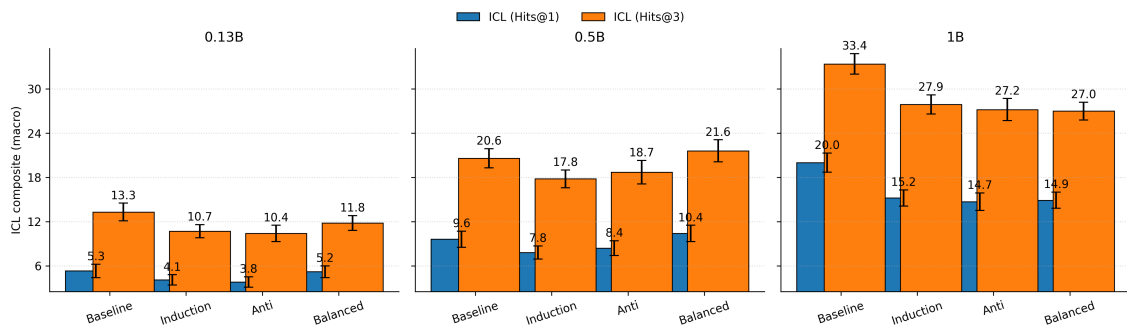


Figure E.4: Function-probe suite of Todd et al. (2024) - ICL Composite (macro). Three panels (0.13B, 0.5B, 1B). For each model, bars compare *ICL (HITS@1)* vs *ICL (HITS@3)* across four regimes (Baseline, Induction, Anti, Balanced). Error bars show ± 1 standard deviation

Decision-Rule Sensitivity - HITS@ k ($k \in \{1, 3\}$): Following common practice for function-probe tasks in the Todd et al. (2024) suite, which reports HITS@3 (top-3 token accuracy), I compare my primary decision rule (HITS@1) with HITS@3. ICL performance is summarised in Figure E.4, and per-task HITS@3 accuracies are listed in Table E.7. While HITS@3 increases absolute scores across the board, the relative ordering and gaps between variants remain effectively unchanged.

	Baseline			Induction		
	0.13B	0.5B	1B	0.13B	0.5B	1B
alphabetically_first-3	8.8 ± 0.8	15.3 ± 0.5	31.0 ± 0.5	8.6 ± 1.3	12.7 ± 0.3	23.0 ± 0.8
alphabetically_first-5	8.6 ± 1.3	13.0 ± 0.5	20.7 ± 1.5	8.3 ± 0.9	12.1 ± 1.0	16.4 ± 0.6
alphabetically_last-3	7.3 ± 0.9	17.4 ± 0.7	30.3 ± 1.0	6.3 ± 0.7	14.3 ± 0.5	24.5 ± 1.1
alphabetically_last-5	5.8 ± 0.8	12.6 ± 1.7	18.6 ± 1.1	4.9 ± 0.8	11.4 ± 0.8	14.6 ± 0.7
capitalize	17.8 ± 1.1	39.7 ± 1.6	73.8 ± 1.4	10.6 ± 0.4	28.5 ± 0.9	57.7 ± 1.3
capitalize_first_letter	25.6 ± 1.6	34.7 ± 1.8	55.2 ± 2.0	17.1 ± 1.1	30.7 ± 1.7	40.1 ± 1.6
capitalize_last_letter	14.1 ± 1.7	25.6 ± 1.7	20.4 ± 2.8	20.0 ± 1.1	18.8 ± 1.9	23.0 ± 2.4
choose_first-of-3	18.3 ± 1.9	38.6 ± 1.9	83.6 ± 0.6	12.9 ± 1.0	27.8 ± 1.7	66.6 ± 1.6
choose_first-of-5	17.1 ± 1.7	30.6 ± 1.9	74.1 ± 1.1	11.3 ± 0.7	24.4 ± 1.6	56.2 ± 1.3
choose_last-of-3	5.1 ± 0.4	8.0 ± 0.6	11.0 ± 1.0	4.5 ± 0.9	8.7 ± 0.7	11.5 ± 1.3
choose_last-of-5	4.0 ± 0.6	7.1 ± 0.7	9.3 ± 1.1	4.9 ± 0.6	7.0 ± 0.8	10.3 ± 0.7
choose_middle-of-3	4.8 ± 1.1	8.7 ± 0.7	12.5 ± 1.0	5.5 ± 0.4	7.7 ± 0.7	13.1 ± 0.7
choose_middle-of-5	4.6 ± 0.7	7.6 ± 0.5	7.7 ± 0.8	4.5 ± 0.2	6.2 ± 0.8	7.4 ± 0.9
lowercase_first_letter	19.4 ± 2.0	28.7 ± 0.9	58.6 ± 2.3	11.4 ± 1.0	22.1 ± 1.4	49.5 ± 1.9
lowercase_last_letter	28.0 ± 1.4	19.4 ± 1.1	23.5 ± 1.2	8.7 ± 1.2	23.1 ± 0.6	26.2 ± 1.3
next_capital_letter	15.0 ± 0.5	13.2 ± 1.2	12.5 ± 0.7	13.2 ± 0.9	12.4 ± 1.8	11.7 ± 1.4
next_item	8.5 ± 2.2	18.7 ± 3.9	29.5 ± 3.2	9.0 ± 1.9	15.8 ± 1.9	23.4 ± 0.8
prev_item	8.4 ± 0.8	15.1 ± 1.4	24.7 ± 2.4	7.6 ± 1.2	14.9 ± 2.1	17.3 ± 1.3
word_length	31.0 ± 2.2	36.5 ± 2.1	37.6 ± 0.6	33.4 ± 1.4	40.2 ± 2.2	38.4 ± 2.2
ICL composite (macro) ↑	13.3 ± 1.2	20.6 ± 1.3	33.4 ± 1.4	10.7 ± 0.9	17.8 ± 1.2	27.9 ± 1.3

Table E.7: Function-probe suite of Todd et al. (2024) under decision-rule sensitivity: HTS@3 accuracy on 10-shot prompts, reported as mean±std across three seeds, for 0.13B, 0.5B, and 1B, comparing Baseline, Induction, Anti, and Balanced curricula.

	Anti-induction			Balanced		
	0.13B	0.5B	1B	0.13B	0.5B	1B
alphabetically_first_3	7.1 $\pm$ 0.3	13.7 $\pm$ 0.8	22.6 $\pm$ 1.4	9.3 $\pm$ 1.2	18.6 $\pm$ 0.6	24.1 $\pm$ 0.8
alphabetically_first_5	7.4 $\pm$ 0.7	13.6 $\pm$ 1.3	15.5 $\pm$ 1.3	7.8 $\pm$ 1.3	13.3 $\pm$ 0.6	15.9 $\pm$ 0.8
alphabetically_last_3	5.7 $\pm$ 0.7	14.6 $\pm$ 1.5	22.3 $\pm$ 1.5	8.9 $\pm$ 0.3	19.1 $\pm$ 1.1	24.0 $\pm$ 0.9
alphabetically_last_5	4.5 $\pm$ 0.5	10.9 $\pm$ 1.4	14.0 $\pm$ 1.0	5.6 $\pm$ 0.8	12.9 $\pm$ 0.9	15.6 $\pm$ 1.0
capitalize	11.5 $\pm$ 1.2	28.9 $\pm$ 1.3	60.6 $\pm$ 0.8	14.0 $\pm$ 2.0	32.2 $\pm$ 1.4	56.0 $\pm$ 0.8
capitalize_first_letter	19.1 $\pm$ 1.5	29.9 $\pm$ 1.7	43.5 $\pm$ 2.1	22.3 $\pm$ 1.7	29.6 $\pm$ 1.3	42.0 $\pm$ 2.1
capitalize_last_letter	20.8 $\pm$ 2.1	23.2 $\pm$ 2.8	17.2 $\pm$ 1.8	15.6 $\pm$ 0.8	15.9 $\pm$ 1.7	18.6 $\pm$ 2.7
choose_first_of_3	9.8 $\pm$ 1.1	28.3 $\pm$ 1.2	60.7 $\pm$ 0.9	19.5 $\pm$ 1.1	50.3 $\pm$ 2.1	67.7 $\pm$ 1.0
choose_first_of_5	8.0 $\pm$ 1.3	26.3 $\pm$ 1.7	48.3 $\pm$ 1.5	15.4 $\pm$ 1.2	41.5 $\pm$ 1.8	56.7 $\pm$ 1.8
choose_last_of_3	3.9 $\pm$ 0.5	7.9 $\pm$ 0.6	11.6 $\pm$ 0.6	4.0 $\pm$ 0.6	8.7 $\pm$ 1.2	12.4 $\pm$ 0.4
choose_last_of_5	3.5 $\pm$ 0.6	7.6 $\pm$ 0.8	10.1 $\pm$ 0.6	4.1 $\pm$ 0.7	6.6 $\pm$ 0.4	10.6 $\pm$ 1.0
choose_middle_of_3	4.0 $\pm$ 0.2	7.8 $\pm$ 0.4	10.3 $\pm$ 0.7	4.4 $\pm$ 0.6	8.1 $\pm$ 1.1	11.4 $\pm$ 0.7
choose_middle_of_5	4.5 $\pm$ 0.9	7.4 $\pm$ 1.1	8.4 $\pm$ 0.7	4.2 $\pm$ 0.8	6.5 $\pm$ 0.9	9.3 $\pm$ 1.0
lowercase_first_letter	12.4 $\pm$ 1.7	22.8 $\pm$ 3.2	45.1 $\pm$ 2.2	5.5 $\pm$ 0.3	30.5 $\pm$ 1.6	38.7 $\pm$ 0.5
lowercase_last_letter	20.2 $\pm$ 1.1	24.0 $\pm$ 0.9	29.7 $\pm$ 0.9	13.2 $\pm$ 1.2	26.8 $\pm$ 2.0	22.8 $\pm$ 1.8
next_capital_letter	16.0 $\pm$ 1.0	13.7 $\pm$ 1.4	10.9 $\pm$ 2.0	12.5 $\pm$ 0.8	11.1 $\pm$ 1.4	13.3 $\pm$ 1.5
next_item	5.8 $\pm$ 2.0	18.2 $\pm$ 3.4	25.4 $\pm$ 1.9	13.5 $\pm$ 1.2	20.5 $\pm$ 3.0	20.3 $\pm$ 2.1
prev_item	5.3 $\pm$ 1.8	17.0 $\pm$ 2.9	20.1 $\pm$ 3.7	13.8 $\pm$ 0.8	17.1 $\pm$ 2.1	17.6 $\pm$ 0.8
word_length	28.5 $\pm$ 1.1	39.9 $\pm$ 1.9	40.0 $\pm$ 3.2	31.5 $\pm$ 1.0	41.1 $\pm$ 2.4	36.6 $\pm$ 1.1
ICL composite (macro) $\uparrow$	10.4 $\pm$ 1.1	18.7 $\pm$ 1.6	27.2 $\pm$ 1.5	11.8 $\pm$ 1.0	21.6 $\pm$ 1.5	27.0 $\pm$ 1.2

Table E.7 (continued): Anti-induction and Balanced results.

E.5 Supplementary head-ablation results

Section 7.1.1 defines load-bearing induction through the targeted-versus-random ablation comparison, and Section 7.3.3 reports the corresponding main results. This appendix provides the implementation details and per-task supplementary results for that causal-reliance analysis.

Selecting induction heads. For each model and curriculum, I compute the per-head induction score defined in Section 7.1.1 and select the top 2% of heads per layer, with the one-head-per-layer floor defined by the localisation rule.

Ablation mechanism: value-stream zeroing. At inference, for a chosen set of heads $\mathcal{S}_\ell$ in layer ℓ , I zero their value-stream contribution before the output projection:

$$\tilde{Z}^{(\ell)} = [QKV^{(1)} \parallel \dots \parallel \underbrace{0}_{h \in \mathcal{S}_\ell} \parallel \dots \parallel QKV^{(H)}], \quad \text{attn_out}^{(\ell)} = \tilde{Z}^{(\ell)} W_O^{(\ell)}.$$

Queries, keys, and softmax weights are unchanged; only the selected heads’ post-attention value vectors are set to zero. This avoids softmax renormalisation artefacts and isolates the contribution of the selected heads’ value streams. I compare two ablation conditions:

- **Induction-head ablation:** zero the per-layer top-2% induction heads defined above.
- **Random-head ablation:** zero the same number of uniformly random heads per layer.

I evaluate the same function-probe prompts, shots, and metrics used for the causal-reliance result in Section 7.3.3. For each model–curriculum pair, I report the clean score and the percent change under each ablation, computed using Equation 7.1. Thus, Δ_{induct} applies Equation 7.1 to induction-head ablation, and Δ_{rand} applies the same equation to matched random-head ablation.

Figure E.5 shows the ICL composite for clean vs. ablations across scales and curricula; per-task deltas are in Tables E.8, E.9, E.10, and E.11.

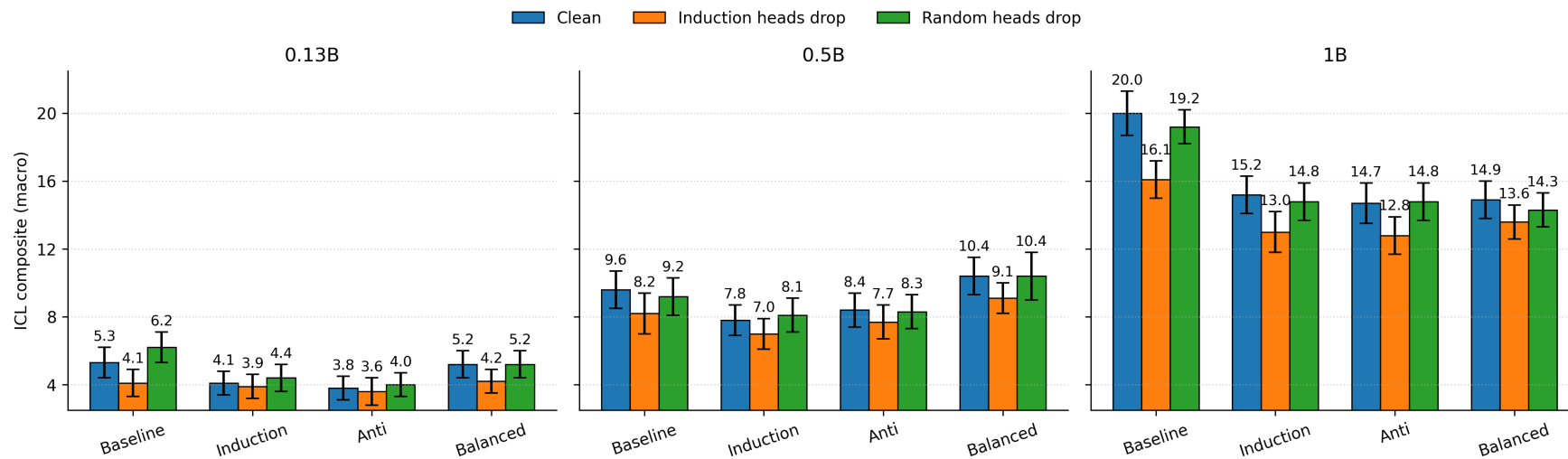


Figure E.5: Function-probe ICL composite under clean evaluation, selected induction-head ablation, and matched random-head ablation. The three panels show 0.13B, 0.5B, and 1B models across Baseline, Induction, Anti-Induction, and Balanced curricula; error bars are ± 1 standard deviation. The landscape layout enlarges labels and error bars for readability.

	0.13B				0.5B			
	Clean	↓Induct	Hd 2%	↓Rand	Clean	↓Induct	Hd 2%	↓Rand
alphabetically_first_3	4.86 ± 0.62	3.74 ± 0.66	5.69 ± 0.37	9.40 ± 0.57	8.86 ± 0.56	8.83 ± 0.58		
alphabetically_first_5	3.97 ± 0.57	3.43 ± 0.52	5.40 ± 0.94	6.71 ± 0.83	7.74 ± 0.66	6.69 ± 1.29		
alphabetically_last_3	3.37 ± 0.55	2.74 ± 0.34	4.69 ± 0.71	10.23 ± 0.88	8.40 ± 0.55	9.17 ± 0.77		
alphabetically_last_5	2.51 ± 0.75	2.09 ± 0.37	3.80 ± 0.78	6.03 ± 1.30	6.11 ± 1.36	6.06 ± 0.34		
capitalize	8.00 ± 1.03	7.02 ± 0.62	9.93 ± 0.83	20.70 ± 1.40	13.19 ± 1.17	15.09 ± 1.61		
capitalize_first_letter	10.07 ± 1.18	7.89 ± 0.98	13.30 ± 1.15	12.53 ± 1.77	11.05 ± 1.70	10.88 ± 1.36		
capitalize_last_letter	4.84 ± 1.27	3.65 ± 0.94	5.19 ± 0.70	9.65 ± 1.05	8.98 ± 0.92	9.40 ± 1.43		
choose_first_of_3	10.34 ± 2.29	6.86 ± 0.74	11.86 ± 0.90	25.46 ± 1.76	20.11 ± 1.77	23.29 ± 1.34		
choose_first_of_5	8.66 ± 1.30	5.31 ± 1.01	10.43 ± 1.15	19.66 ± 1.94	17.63 ± 2.75	18.14 ± 2.56		
choose_last_of_3	2.03 ± 0.42	1.86 ± 0.29	2.14 ± 0.40	3.17 ± 0.16	3.17 ± 1.29	2.86 ± 0.44		
choose_last_of_5	1.51 ± 0.33	1.57 ± 0.58	2.00 ± 0.53	2.86 ± 0.61	2.46 ± 0.60	2.31 ± 0.38		
choose_middle_of_3	1.69 ± 0.64	1.49 ± 0.16	1.83 ± 0.60	3.34 ± 0.55	3.31 ± 0.48	3.06 ± 0.47		
choose_middle_of_5	1.63 ± 0.30	1.63 ± 0.46	2.00 ± 0.73	2.97 ± 0.64	3.29 ± 0.48	2.69 ± 0.52		
lowercase_first_letter	5.51 ± 0.77	4.74 ± 0.99	7.72 ± 1.05	8.42 ± 0.86	7.26 ± 1.02	7.72 ± 1.11		
lowercase_last_letter	11.09 ± 0.85	7.51 ± 1.80	8.49 ± 1.63	7.72 ± 0.73	7.37 ± 0.98	7.54 ± 1.19		
next_capital_letter	4.88 ± 1.09	3.19 ± 0.98	6.42 ± 0.59	4.21 ± 0.94	3.44 ± 1.56	4.32 ± 0.20		
next_item	3.42 ± 2.17	2.78 ± 2.03	5.19 ± 1.22	8.61 ± 2.48	5.95 ± 2.26	13.16 ± 1.51		
prev_item	2.53 ± 0.78	2.03 ± 1.44	3.67 ± 1.51	7.72 ± 1.64	5.32 ± 1.15	11.39 ± 1.85		
word_length	9.89 ± 0.99	9.19 ± 1.06	8.74 ± 1.05	13.33 ± 0.94	11.96 ± 0.67	12.70 ± 1.76		
ICL Composite (macro)	5.3 ± 0.9	4.1 ± 0.8	6.2 ± 0.9	9.6 ± 1.1	8.2 ± 1.2	9.2 ± 1.1		

Table E.8: **Function-probe suite of Todd et al. (2024)** (10-shot HITS@1): per-task accuracy (mean±std over three seeds) for **Baseline** curriculum. I ablate the top-2% induction heads (↓Induct Hd 2%) or an equal number of random heads (↓Rand).

	1B		
	Clean	↓Induct Hd 2%	↓Rand
alphabetically_first-3	19.71 ± 0.90	15.00 ± 0.86	19.51 ± 0.96
alphabetically_first-5	12.37 ± 1.41	11.03 ± 0.93	12.31 ± 1.14
alphabetically_last-3	20.80 ± 0.64	16.31 ± 1.23	18.80 ± 1.01
alphabetically_last-5	9.89 ± 0.82	9.20 ± 0.87	9.83 ± 0.76
capitalize	54.81 ± 2.05	36.74 ± 2.61	52.42 ± 2.48
capitalize_first_letter	28.56 ± 1.08	22.70 ± 0.52	24.74 ± 0.92
capitalize_last_letter	8.28 ± 1.21	6.84 ± 1.27	7.61 ± 1.14
choose_first-of-3	69.37 ± 1.77	49.20 ± 2.03	68.80 ± 0.61
choose_first-of-5	55.51 ± 1.26	39.20 ± 2.13	53.71 ± 0.83
choose_last-of-3	4.40 ± 0.98	5.11 ± 1.18	4.23 ± 0.88
choose_last-of-5	3.89 ± 1.14	4.06 ± 0.65	3.63 ± 0.31
choose_middle-of-3	4.23 ± 0.48	4.94 ± 0.52	4.37 ± 0.59
choose_middle-of-5	2.71 ± 0.36	3.23 ± 0.56	2.89 ± 0.34
lowercase_first_letter	28.53 ± 2.16	24.63 ± 1.69	26.60 ± 1.94
lowercase_last_letter	10.46 ± 1.09	10.32 ± 1.00	8.81 ± 1.20
next_capital_letter	2.42 ± 0.81	1.86 ± 0.24	2.74 ± 0.40
next_item	16.84 ± 2.13	17.09 ± 1.10	16.08 ± 0.57
prev_item	15.06 ± 2.30	15.82 ± 1.10	14.68 ± 1.13
word_length	12.77 ± 1.11	13.12 ± 0.68	12.07 ± 1.87
ICL Composite (macro)	20.0 ± 1.3	16.1 ± 1.1	19.2 ± 1.0

Table E.8 (continued): 1B results.

	0.13B				0.5B			
	Clean	↓Induct	Hd 2%	↓Rand	Clean	↓Induct	Hd 2%	↓Rand
alphabetically_first_3	4.63 ± 0.57	4.43 ± 0.93	4.14 ± 0.53	7.34 ± 0.33	5.89 ± 0.57	7.03 ± 0.48		
alphabetically_first_5	4.23 ± 0.79	4.89 ± 0.42	4.57 ± 0.44	6.51 ± 1.03	6.46 ± 0.74	6.94 ± 0.80		
alphabetically_last_3	2.43 ± 0.54	2.31 ± 0.40	2.54 ± 0.55	7.69 ± 0.16	7.29 ± 0.77	7.83 ± 0.56		
alphabetically_last_5	1.60 ± 0.27	1.31 ± 0.48	1.89 ± 0.12	5.49 ± 0.39	5.40 ± 0.99	6.06 ± 0.61		
capitalize	3.33 ± 0.28	3.75 ± 1.09	4.56 ± 0.25	13.19 ± 1.27	12.00 ± 1.28	14.84 ± 1.24		
capitalize_first_letter	5.33 ± 1.00	5.40 ± 1.11	5.26 ± 0.62	13.40 ± 1.65	11.40 ± 1.16	14.14 ± 1.27		
capitalize_last_letter	8.63 ± 1.12	10.88 ± 0.66	6.81 ± 0.70	7.51 ± 1.75	7.09 ± 0.88	5.93 ± 0.95		
choose_first_of_3	6.14 ± 0.89	4.80 ± 0.85	6.11 ± 0.67	19.03 ± 1.72	15.14 ± 1.55	20.06 ± 2.31		
choose_first_of_5	4.86 ± 0.73	4.23 ± 0.39	5.26 ± 0.58	15.31 ± 0.91	11.74 ± 1.82	16.31 ± 1.40		
choose_last_of_3	1.43 ± 0.44	1.63 ± 0.34	1.94 ± 0.28	2.83 ± 0.26	2.54 ± 1.13	2.66 ± 0.52		
choose_last_of_5	1.74 ± 0.47	1.63 ± 0.55	2.26 ± 0.44	2.37 ± 0.41	1.94 ± 0.16	2.69 ± 0.63		
choose_middle_of_3	2.11 ± 0.80	1.86 ± 0.53	1.89 ± 0.51	2.23 ± 0.26	2.09 ± 0.41	2.37 ± 0.30		
choose_middle_of_5	1.60 ± 0.23	1.63 ± 0.49	1.60 ± 0.53	2.09 ± 0.28	1.66 ± 0.46	1.86 ± 0.48		
lowercase_first_letter	4.14 ± 0.92	2.63 ± 0.30	6.00 ± 1.31	6.07 ± 1.23	5.61 ± 0.72	7.26 ± 0.75		
lowercase_last_letter	3.61 ± 0.67	2.84 ± 0.34	3.65 ± 1.16	8.04 ± 0.73	7.86 ± 1.48	8.18 ± 1.72		
next_capital_letter	3.86 ± 0.33	4.28 ± 1.08	3.82 ± 0.78	4.18 ± 0.96	4.04 ± 0.83	3.44 ± 0.64		
next_item	2.91 ± 1.93	2.66 ± 1.58	4.30 ± 2.07	5.70 ± 1.34	6.46 ± 1.44	7.72 ± 1.76		
prev_item	2.03 ± 0.83	2.78 ± 1.31	3.04 ± 1.04	5.95 ± 0.72	6.71 ± 0.96	6.20 ± 1.70		
word_length	13.40 ± 0.83	10.21 ± 0.98	13.86 ± 1.67	14.00 ± 0.97	11.79 ± 0.57	12.49 ± 1.31		
ICL Composite (macro)	4.1 ± 0.7	3.9 ± 0.7	4.4 ± 0.8	7.8 ± 0.9	7.0 ± 0.9	8.1 ± 1.0		

Table E.9: **Function-probe suite of Todd et al. (2024)** (10-shot HITS@1): per-task accuracy (mean±std over three seeds) for **Induction** curriculum. I ablate the top-2% induction heads (↓Induct Hd 2%) or an equal number of random heads (↓Rand).

	1B		
	Clean	↓Induct Hd 2%	↓Rand
alphabetically_first-3	15.11 ± 0.89	12.94 ± 0.81	15.09 ± 0.91
alphabetically_first-5	10.89 ± 0.87	9.09 ± 0.65	9.63 ± 1.21
alphabetically_last-3	15.77 ± 0.96	12.31 ± 0.64	15.94 ± 0.97
alphabetically_last-5	8.51 ± 0.33	7.51 ± 1.21	8.97 ± 0.62
capitalize	33.37 ± 1.31	25.93 ± 0.72	31.30 ± 1.15
capitalize_first_letter	13.68 ± 1.16	11.58 ± 1.41	14.28 ± 1.23
capitalize_last_letter	8.63 ± 0.85	9.33 ± 1.17	7.30 ± 0.90
choose_first-of-3	52.37 ± 2.00	41.03 ± 2.98	52.89 ± 1.19
choose_first-of-5	42.63 ± 3.05	33.89 ± 3.08	43.63 ± 1.43
choose_last-of-3	5.40 ± 0.92	5.77 ± 1.38	5.63 ± 0.96
choose_last-of-5	4.57 ± 0.58	4.86 ± 0.61	4.03 ± 0.62
choose_middle-of-3	6.00 ± 1.03	5.51 ± 0.36	5.51 ± 0.64
choose_middle-of-5	3.09 ± 0.62	3.66 ± 0.52	3.03 ± 0.63
lowercase_first_letter	20.11 ± 1.56	17.05 ± 1.46	18.21 ± 1.22
lowercase_last_letter	9.61 ± 0.96	9.26 ± 0.44	7.68 ± 1.43
next_capital_letter	3.58 ± 0.72	3.47 ± 0.68	3.09 ± 0.36
next_item	11.77 ± 1.23	11.14 ± 2.03	12.91 ± 2.13
prev_item	10.25 ± 1.04	9.62 ± 2.16	9.49 ± 1.34
word_length	13.33 ± 0.84	12.18 ± 1.10	13.26 ± 1.19
ICL Composite (macro)	15.2 ± 1.1	13.0 ± 1.2	14.8 ± 1.1

Table E.9 (continued): 1B results.

	0.13B				0.5B			
	Clean	↓Induct	Hd 2%	↓Rand	Clean	↓Induct	Hd 2%	↓Rand
alphabetically_first_3	3.34 ± 0.55	2.83 ± 0.50	3.57 ± 0.45	8.46 ± 0.88	7.40 ± 0.26	8.00 ± 0.51		
alphabetically_first_5	2.83 ± 0.68	2.57 ± 0.34	2.71 ± 0.30	7.37 ± 0.63	7.69 ± 0.23	7.80 ± 1.03		
alphabetically_last_3	1.94 ± 0.60	2.20 ± 0.30	2.60 ± 0.40	7.97 ± 0.87	7.71 ± 0.61	7.63 ± 0.58		
alphabetically_last_5	1.77 ± 0.36	1.49 ± 0.58	2.09 ± 0.59	6.06 ± 0.79	5.89 ± 0.91	6.94 ± 0.59		
capitalize	3.72 ± 0.15	5.33 ± 1.26	4.67 ± 0.75	13.47 ± 1.33	11.16 ± 1.34	13.30 ± 1.62		
capitalize_first_letter	5.19 ± 0.32	6.11 ± 2.06	5.58 ± 0.45	11.16 ± 1.21	10.56 ± 1.75	10.67 ± 1.33		
capitalize_last_letter	9.09 ± 1.39	6.74 ± 1.56	7.93 ± 1.13	9.30 ± 1.16	8.39 ± 0.89	8.04 ± 1.22		
choose_first_of_3	4.11 ± 0.72	3.60 ± 0.21	3.83 ± 0.36	19.06 ± 1.75	16.69 ± 1.56	19.60 ± 1.64		
choose_first_of_5	3.63 ± 0.67	2.69 ± 0.40	3.17 ± 0.38	16.20 ± 1.49	14.43 ± 1.66	17.49 ± 1.29		
choose_last_of_3	1.40 ± 0.51	0.94 ± 0.19	0.94 ± 0.13	2.97 ± 0.52	3.34 ± 0.50	3.46 ± 1.11		
choose_last_of_5	1.09 ± 0.37	0.97 ± 0.33	1.31 ± 0.31	2.86 ± 0.52	2.66 ± 0.26	2.51 ± 0.47		
choose_middle_of_3	1.34 ± 0.52	1.31 ± 0.70	1.43 ± 0.23	3.09 ± 0.24	2.63 ± 0.52	2.94 ± 0.52		
choose_middle_of_5	1.66 ± 0.36	1.71 ± 0.44	1.57 ± 0.39	2.86 ± 0.69	2.86 ± 0.30	1.94 ± 0.52		
lowercase_first_letter	4.67 ± 0.74	3.61 ± 0.97	4.98 ± 1.01	6.53 ± 0.59	6.56 ± 1.17	7.58 ± 0.60		
lowercase_last_letter	7.37 ± 1.15	5.47 ± 0.64	8.18 ± 1.45	8.88 ± 0.97	9.26 ± 0.73	8.04 ± 1.58		
next_capital_letter	4.67 ± 0.79	5.05 ± 1.24	5.09 ± 0.92	4.14 ± 0.71	3.65 ± 0.57	4.04 ± 0.55		
next_item	1.27 ± 0.90	2.15 ± 0.72	2.41 ± 1.37	7.85 ± 1.06	6.71 ± 2.71	6.71 ± 0.72		
prev_item	1.39 ± 0.53	2.28 ± 1.65	1.77 ± 0.28	6.58 ± 2.52	5.70 ± 1.85	7.09 ± 0.83		
word_length	12.00 ± 1.29	11.05 ± 1.20	11.40 ± 1.59	14.32 ± 0.58	13.33 ± 0.75	14.18 ± 1.68		
ICL Composite (macro)	3.8 ± 0.7	3.6 ± 0.8	4.0 ± 0.7	8.4 ± 1.0	7.7 ± 1.0	8.3 ± 1.0		

Table E.10: **Function-probe suite of Todd et al. (2024)** (10-shot HITS@1): per-task accuracy (mean±std over three seeds) for **Anti-Induction** curriculum. I ablate the top-2% induction heads (↓Induct Hd 2%) or an equal number of random heads (↓Rand).

	1B		
	Clean	↓Induct Hd 2%	↓Rand
alphabetically_first_3	14.43 ± 1.47	11.34 ± 1.13	14.49 ± 1.33
alphabetically_first_5	8.43 ± 1.15	7.97 ± 0.76	9.11 ± 0.80
alphabetically_last_3	13.26 ± 0.77	10.40 ± 0.75	13.80 ± 0.46
alphabetically_last_5	7.80 ± 0.48	7.14 ± 0.83	8.29 ± 0.42
capitalize	39.23 ± 2.71	34.18 ± 1.46	37.40 ± 0.83
capitalize_first_letter	17.44 ± 0.84	17.65 ± 2.57	20.98 ± 1.51
capitalize_last_letter	7.26 ± 1.49	6.67 ± 0.76	6.32 ± 1.40
choose_first_of_3	46.03 ± 2.04	34.43 ± 1.29	46.31 ± 1.30
choose_first_of_5	32.69 ± 1.64	23.97 ± 0.53	33.71 ± 1.21
choose_last_of_3	5.00 ± 0.51	5.11 ± 0.86	5.03 ± 1.18
choose_last_of_5	5.63 ± 0.47	5.06 ± 0.52	4.86 ± 0.92
choose_middle_of_3	4.97 ± 0.27	4.97 ± 0.31	4.83 ± 0.93
choose_middle_of_5	3.89 ± 0.73	3.71 ± 0.27	3.34 ± 0.89
lowercase_first_letter	20.56 ± 1.11	18.74 ± 1.52	18.98 ± 1.86
lowercase_last_letter	13.26 ± 0.47	12.46 ± 1.75	12.67 ± 1.57
next_capital_letter	3.05 ± 0.91	3.19 ± 0.60	2.98 ± 0.79
next_item	11.77 ± 2.48	12.41 ± 2.31	13.04 ± 1.15
prev_item	11.39 ± 2.45	12.03 ± 1.00	10.89 ± 1.44
word_length	13.58 ± 1.60	12.49 ± 1.83	13.37 ± 1.69
ICL Composite (macro)	14.7 ± 1.2	12.8 ± 1.1	14.8 ± 1.1

Table E.10 (continued): 1B results.

	0.13B			0.5B		
	Clean	↓Induct Hd 2%	↓Rand	Clean	↓Induct Hd 2%	↓Rand
alphabetically_first_3	4.83 ± 0.62	3.17 ± 0.37	4.71 ± 0.80	11.54 ± 0.60	9.54 ± 0.47	11.26 ± 1.04
alphabetically_first_5	3.94 ± 0.97	3.14 ± 0.83	4.26 ± 0.93	8.23 ± 0.78	6.80 ± 0.67	8.11 ± 1.16
alphabetically_last_3	3.94 ± 0.47	2.00 ± 0.53	4.74 ± 0.43	10.89 ± 0.50	9.71 ± 0.66	11.49 ± 0.16
alphabetically_last_5	2.51 ± 0.77	1.34 ± 0.22	2.14 ± 0.35	6.71 ± 0.55	6.54 ± 0.64	7.09 ± 0.41
capitalize	6.00 ± 1.63	5.65 ± 1.06	7.09 ± 1.12	14.67 ± 1.30	14.49 ± 1.15	12.77 ± 1.99
capitalize_first_letter	8.88 ± 0.66	7.75 ± 1.39	9.44 ± 0.96	10.00 ± 1.07	11.65 ± 1.76	10.98 ± 2.45
capitalize_last_letter	5.51 ± 0.67	5.65 ± 0.57	4.04 ± 0.92	5.75 ± 1.38	7.37 ± 0.67	5.33 ± 1.26
choose_first_of_3	11.29 ± 0.71	4.37 ± 0.46	11.71 ± 0.69	35.20 ± 1.70	23.49 ± 1.25	35.51 ± 2.16
choose_first_of_5	7.89 ± 1.01	3.43 ± 0.34	7.43 ± 0.77	28.20 ± 2.41	19.31 ± 0.53	27.74 ± 2.46
choose_last_of_3	1.69 ± 0.34	1.20 ± 0.16	1.37 ± 0.44	3.00 ± 0.53	4.00 ± 0.94	3.34 ± 0.70
choose_last_of_5	1.46 ± 0.33	1.14 ± 0.30	1.40 ± 0.42	2.63 ± 0.36	3.09 ± 0.47	2.89 ± 0.62
choose_middle_of_3	1.71 ± 0.66	1.20 ± 0.30	1.46 ± 0.23	3.40 ± 0.69	3.66 ± 0.58	3.11 ± 1.00
choose_middle_of_5	1.74 ± 0.44	1.40 ± 0.47	1.69 ± 0.29	2.34 ± 0.63	3.51 ± 0.22	2.71 ± 1.04
lowercase_first_letter	2.25 ± 0.57	4.14 ± 0.90	2.39 ± 0.69	8.84 ± 0.73	5.75 ± 0.44	9.44 ± 1.81
lowercase_last_letter	7.82 ± 1.10	9.79 ± 1.18	7.19 ± 0.71	10.88 ± 1.53	8.84 ± 1.68	8.67 ± 1.72
next_capital_letter	4.81 ± 0.91	3.96 ± 0.69	4.88 ± 0.88	3.47 ± 0.89	3.33 ± 0.62	4.07 ± 0.81
next_item	6.33 ± 0.78	5.44 ± 1.82	7.85 ± 1.46	8.99 ± 1.64	8.73 ± 1.51	10.38 ± 2.85
prev_item	5.70 ± 1.27	5.19 ± 1.13	7.97 ± 1.65	8.35 ± 1.22	9.49 ± 1.73	8.61 ± 2.39
word_length	10.81 ± 1.09	10.14 ± 1.33	7.89 ± 0.76	14.74 ± 1.81	12.95 ± 1.16	13.51 ± 1.22
ICL Composite (macro)	5.2 ± 0.8	4.2 ± 0.7	5.2 ± 0.8	10.4 ± 1.1	9.1 ± 0.9	10.4 ± 1.4

Table E.11: **Function-probe suite of Todd et al. (2024)** (10-shot HITS@1): per-task accuracy (mean±std over three seeds) for **Balanced** curriculum. I ablate the top-2% induction heads (↓Induct Hd 2%) or an equal number of random heads (↓Rand).

	1B		
	Clean	↓Induct Hd 2%	↓Rand
alphabetically_first_3	15.31 ± 0.96	13.66 ± 0.58	14.83 ± 0.68
alphabetically_first_5	10.20 ± 0.70	9.11 ± 0.69	10.26 ± 0.72
alphabetically_last_3	15.34 ± 1.35	13.91 ± 0.46	15.20 ± 0.78
alphabetically_last_5	9.31 ± 0.43	9.37 ± 0.63	9.63 ± 0.50
capitalize	33.65 ± 2.14	27.93 ± 1.35	31.82 ± 1.33
capitalize_first_letter	20.39 ± 1.17	18.84 ± 1.32	19.26 ± 1.54
capitalize_last_letter	6.67 ± 0.66	5.68 ± 1.68	6.11 ± 0.65
choose_first_of_3	54.09 ± 1.70	46.29 ± 1.12	50.03 ± 1.59
choose_first_of_5	42.14 ± 1.37	34.14 ± 1.15	39.91 ± 0.82
choose_last_of_3	5.20 ± 0.34	5.86 ± 0.91	5.69 ± 0.69
choose_last_of_5	5.06 ± 0.36	4.97 ± 0.42	5.80 ± 0.44
choose_middle_of_3	4.94 ± 0.70	5.20 ± 0.85	5.34 ± 0.63
choose_middle_of_5	4.66 ± 0.63	4.54 ± 0.86	4.37 ± 0.49
lowercase_first_letter	13.30 ± 0.84	13.61 ± 1.09	13.40 ± 1.56
lowercase_last_letter	8.91 ± 1.18	9.09 ± 2.06	6.56 ± 1.77
next_capital_letter	4.18 ± 1.15	2.56 ± 0.56	3.37 ± 0.74
next_item	9.37 ± 1.70	10.89 ± 1.87	9.87 ± 0.96
prev_item	7.72 ± 2.21	10.51 ± 0.57	7.97 ± 0.96
word_length	12.60 ± 0.72	12.42 ± 1.49	12.91 ± 1.60
ICL Composite (macro)	14.9 ± 1.1	13.6 ± 1.0	14.3 ± 1.0

Table E.11 (continued): 1B results.