A Study of Prompt Tuning on Small Language Models(SLMs):

A Controlled Benchmark and a Lightweight

Instance-Aware Method

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology in Computer Science

by

Narkadamilli Sahith

Roll No: CS2419

under the supervision of

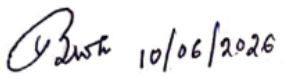
Prof. Ujjwal Bhattacharya
Professor
Computer Vision and Pattern Recognition Unit

Indian Statistical Institute, Kolkata 700108, INDIA

June 2026

CERTIFICATE

This is to certify that the dissertation entitled **"A Study of Prompt Tuning on Small Language Models(SLMs)"** submitted by **Narkadamilli Sahith** (CS2419) to the Indian Statistical Institute, Kolkata, in partial fulfilment of the requirements for the award of the degree of **Master of Technology in Computer Science (with specialization in Data Science)** is a bona fide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.



10/06/2026

Prof. Ujjwal Bhattacharya
Professor,
Computer Vision and Pattern Recognition Unit,
Indian Statistical Institute,
Kolkata 700108, India.

To my family, for their constant support,
and to my supervisor, for the guidance that shaped this work.

Acknowledgements

I am deeply grateful to my supervisor, Prof. Ujjwal Bhattacharya, for the guidance, insight, and encouragement that shaped this dissertation. I thank the faculty and researchers of Computer Vision and Pattern Recognition Unit, Indian Statistical Institute, Kolkata, for fostering a stimulating academic environment, and my peers for the discussions that enriched this work. Finally, I thank my family for their unwavering support throughout this journey.

Abstract

Parameter-efficient fine-tuning (PEFT) adapts a frozen pre-trained language model by training only a small number of additional parameters. Among PEFT approaches, prompt tuning prepends trainable continuous vectors (soft prompts) to the input. A recurring finding in the literature is that prompt tuning is strongly scale dependent: it rivals full fine-tuning on very large models but lags on smaller ones. This dissertation studies prompt tuning specifically in the small-language-model (SLM) regime. We (i) re-implement a representative set of prompt-tuning methods—Prompt Tuning, P-Tuning v2, LoPT, DPT, DePT, ACCEPT, Residual Prompt Tuning, and PARA—within a single controlled harness, enabling a fair head-to-head comparison against full fine-tuning; (ii) propose IA-DePT, a lightweight instance-aware extension of Decomposed Prompt Tuning that conditions the short soft prompt on each input through a small, zero-initialised gate; and (iii) extend the benchmark beyond a single backbone and task, evaluating the full method suite on *six* backbone/task settings that span encoder–decoder (t5-small), encoder-only (BERT-base, RoBERTa-base, ELECTRA-small), and decoder-only (DistilGPT-2) architectures across the GLUE/SuperGLUE tasks RTE, WSC, CB, COPA, WiC, and MRPC. On RTE with t5-small, IA-DePT is the strongest parameter-efficient method in our benchmark (55.6% single-seed accuracy) and improves over its own base, DePT, by 6.5 points (53.6% vs. 47.1%, mean over three seeds) while adding only ≈16.9k parameters—a total trainable footprint of 0.05% of the backbone. Because the gate degrades exactly to DePT at initialisation, the comparison is a clean single-variable ablation. The cross-architecture study shows that the instance gate improves on DePT in five of the six settings on each setting's primary metric (it ties or marginally regresses only on WiC, where every PEFT method sits at chance), so the benefit is broad but not universal. Our analysis characterises the accuracy/parameter trade-offs across method families, the strong effect of task difficulty on the small-model regime, and the role of instance-conditioning, including an honest discussion of why many prompt-tuning methods remain close to the chance baseline at this scale.

Keywords: Parameter-Efficient Fine-Tuning, Prompt Tuning, Small Language Models, Decomposed Prompt Tuning, Instance-Aware Prompting, GLUE, SuperGLUE.

Abbreviations

Abbreviation	Full Form
PEFT	Parameter-Efficient Fine-Tuning
PT	Prompt Tuning
SLM	Small Language Model
LM	Language Model
DPT	Decomposed Prompt Tuning (low-rank product)
DePT	Decomposed Prompt Tuning (prompt + embedding update)
IA-DePT	Instance-Aware DePT (this work)
ACCEPT	Adaptive Codebook for Composite and Efficient Prompt Tuning
PARA	Prompt-Aware Representation Adjustment
LoPT	Low-Rank Prompt Tuning
IDPG	Instance-Dependent Prompt Generation
LoRA	Low-Rank Adaptation
RTE	Recognizing Textual Entailment
WSC	Winograd Schema Challenge
CB	CommitmentBank
COPA	Choice of Plausible Alternatives
WiC	Words in Context
MRPC	Microsoft Research Paraphrase Corpus
MLP	Multi-Layer Perceptron
GELU	Gaussian Error Linear Unit
PQ	Product Quantization
EM	Exact Match
MCC	Matthews Correlation Coefficient

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Adapting large pre-trained language models to downstream tasks by full fine-tuning is expensive: a separate copy of all model parameters must be stored for every task. Parameter-efficient fine-tuning (PEFT) instead freezes the backbone and trains only a small set of extra parameters [12]. Prompt tuning [1] is an appealingly simple PEFT method: a sequence of trainable continuous vectors (a soft prompt) is prepended to the input embeddings while the model is kept frozen. Because only the prompt is stored per task, the marginal cost of a new task is negligible.

A central, well-documented limitation of prompt tuning is scale dependency. Lester et al. [1] show that prompt tuning becomes competitive with full fine-tuning only as model size grows, and a recent survey reiterates model-scale dependency as an open challenge [11]. Most subsequent improvements are nonetheless evaluated on mid-to-large backbones (t5-base and up). The behaviour of these methods on small language models, and which design choices actually help in that regime, remains comparatively under-examined. This gap motivates the present study: rather than chasing the best number on a large model, we fix small backbones and ask which prompt-tuning ideas survive at small scale, and whether they survive *consistently* across architectures and tasks.

1.2 Research questions

RQ1.

Under a single controlled harness on a small LM, how do representative prompt-tuning methods from different design families compare in accuracy and parameter efficiency?

RQ2.

Does instance-conditioning (making the soft prompt depend on the input) improve a strong decomposed prompt-tuning method in the small-model regime?

RQ3.

What accuracy/efficiency trade-offs distinguish the method families (low-rank, decomposition, codebook, reparameterisation, deep prompt, hidden-state adjustment) on an SLM?

RQ4.

Do the answers to RQ1–RQ3 generalise across small backbones (encoder–decoder, encoder-only, decoder-only) and across tasks of differing difficulty?

1.3 Contributions

1. A controlled reproduction and benchmark of nine approaches—full fine-tuning, vanilla Prompt Tuning, P-Tuning v2, LoPT, DPT, DePT, ACCEPT, Residual Prompt Tuning, and PARA—in a single harness, enabling direct comparison across methods previously evaluated under heterogeneous conditions.

2. IA-DePT, a lightweight instance-aware extension of Decomposed Prompt Tuning, in which a zero-initialised gate modulates the soft prompt per input. By construction, gate off = DePT and gate on = IA-DePT, giving a clean single-variable ablation.

3. A *cross-architecture, cross-task* study across six backbone/task settings spanning encoder–decoder, encoder-only, and decoder-only models (Table 4.1), testing how broadly the conclusions and the instance gate generalise.

4. An analysis of accuracy/parameter trade-offs across method families and of when instance-conditioning helps (or does not) for small models, including an honest discussion of the many methods that hover near the chance baseline in this regime and of the strong role played by task difficulty.

1.4 Dissertation structure

Chapter 2 reviews PEFT and a taxonomy of prompt tuning. Chapter 3 describes the unified harness, the reproduced methods, and the proposed IA-DePT. Chapter 4 details the experimental setup, including the six backbone/task settings. Chapter 5 presents the controlled single-task results on RTE, and Chapter 6 reports the cross-architecture, cross-task generalisation study. Chapter 7 concludes and outlines future work.

Chapter 2

Background and Related Work

2.1 Parameter-efficient fine-tuning

PEFT methods adapt a frozen model with few trainable parameters. They are commonly grouped into three families [12]: additive methods that insert new modules (e.g. Adapters [18]); selective methods that update a chosen subset of existing parameters (e.g. BitFit [19]); and reparameterisation methods that express weight updates in a low-rank form (e.g. LoRA [17]). Soft-prompt methods are an additive family that operates at, or near, the input. Their distinctive advantage is storage: a task is captured by a handful of vectors rather than a delta over the full weight matrix, which matters most when many tasks must coexist.

2.2 A taxonomy of prompt tuning

Following the recent survey of Li and Collier [11], prompt-tuning methods can be organised into direct prompt learning (optimising a prompt for a single task) and prompt transfer learning (transferring prompts across tasks). Within direct learning we distinguish the design families that this dissertation benchmarks.

General optimisation. Vanilla Prompt Tuning [1] optimises a prepended soft prompt directly. Prefix-Tuning [2] and P-Tuning v2 [3] extend trainable vectors to every layer (deep prompts), trading parameters for capacity.

Low-rank / decomposition. These reduce the prompt's parameter count. DPT [4] reparameterises the prompt as a product of two low-rank matrices, motivated by the observation that soft prompts have a low intrinsic rank; LoPT [9] similarly imposes a

low-rank structure on the whole prompt. DePT [5] decomposes the prompt into a shorter soft prompt plus a low-rank update added to the (frozen) word embeddings, shortening the input sequence and thus reducing the quadratic attention cost.

Codebook / quantization. ACCEPT [6] represents sub-prompts as soft combinations of shared product-quantization codewords, so the codebook size is decoupled from prompt length.

Reparameterisation for optimisation. Residual Prompt Tuning [7] passes the prompt through a shallow MLP with a residual connection, improving convergence—of particular relevance to small models, where optimisation is a key difficulty.

Hidden-state adjustment. Rather than adding tokens, PARA [8] inserts a per-layer prompt-aware generator that scales the attention and feed-forward hidden states.

Instance-aware prompting. IDPG [10] generates a different prompt for each input via a trainable generator. Our proposed method is related but deliberately lighter (see Section 3.3).

Transfer / multi-task (context). SPoT [13], ATTEMPT [14], MPT [15], and TPT [16] transfer or compose prompts across tasks. These require source-task pre-training and are discussed for completeness but lie outside our single-task benchmark.

2.3 The small-model regime

The works above largely report on t5-base or larger. Because prompt tuning is scale dependent [1, 11], conclusions drawn at larger scales do not necessarily transfer to small models: a static prompt that suffices to steer a 220M-parameter model may exert too little influence on a 14–125M one, and the optimisation landscape is harder when the trainable budget is tiny. This dissertation therefore fixes small backbones and asks which methods and design choices remain effective, and whether they do so consistently across architectures and tasks.

Chapter 3

Methodology

3.1 A unified evaluation harness

All methods are implemented on a single frozen backbone and evaluated on a binary or multi-class classification task. For the encoder–decoder backbone (t5-small, $d = 512$) the task is cast as a text-to-text problem: the input is `"premise: ...hypothesis: ..."` and the target is the string `"0"` or `"1"`; predictions are produced by greedy generation and scored by exact-match accuracy. For the encoder-only and decoder-only backbones the task is cast as sequence classification: a small linear head reads a pooled representation of the (prompt-augmented) final hidden states and is trained with cross-entropy. Every method shares the same tokenizer, sequence length, optimiser family, and evaluation procedure within a setting; only the trainable module differs. This control is what makes the cross-method comparison meaningful, and is itself a contribution given that prior results are spread across heterogeneous backbones and tasks.

Let $E \in \mathbb{R}^{s \times d}$ be the input word embeddings (s tokens, dimension d), and let the backbone parameters θ be frozen throughout. Each method below defines a small trainable module and a rule for composing its output with E before the frozen backbone.

3.2 Reproduced methods

Prompt Tuning (PT). A trainable prompt $P \in \mathbb{R}^{m \times d}$ is prepended: the model sees $[\,P; E\,]$ and we maximise $p_\theta(y \mid [P; E])$ over P [1]. We use $m = 50$ virtual tokens via the PEFT library.

P-Tuning v2 (deep prompt). Trainable key/value vectors are injected at every layer (deep prompts), realised via PEFT prefix-tuning [3, 2]. We use $m = 20$.

LoPT / DPT. The prompt is reparameterised as a low-rank product $P = AB$ with $A \in \mathbb{R}^{m\times r}$, $B \in \mathbb{R}^{r\times d}$, $r \ll d$ [4, 9]. We use $m = 100$, $r = 10$. LoPT and DPT share this low-rank-prompt mechanism; we treat them as one family and report both so the family is represented twice.

DePT. A short prompt $P_s \in \mathbb{R}^{m\times d}$ is combined with a low-rank additive update to the word embeddings, $\Delta E = A_d B_d$:

$$\text{input} = [\, P_s;\ E + \Delta E\,], \qquad A_d \in \mathbb{R}^{s\times r},\ B_d \in \mathbb{R}^{r\times d}. \tag{3.1}$$

B_d is zero-initialised so that $\Delta E = 0$ at the start [5]. We use a short prompt of length $m = 20$ and rank $r = 8$.

ACCEPT. The embedding dimension is split into K subspaces of size $t = d/K$, each with a shared codebook of r codewords. A sub-prompt is a soft (weighted) combination of codewords, $p_i^k = \sum_{j=1}^{r} w_{ij}^k\, c_j^k$, used to build both a prepended prompt (SCPP) and an additive prompt on the embeddings (SCAP). The trainable parameters total $rd + rmK$, independent of prompt length [6]. We use SCPP with $m = 20$, $K = 16$, $r = 20$ and SCAP with $K = 2$, $r = 8$; the SCAP weights are zero-initialised so its contribution starts at zero.

Residual Prompt Tuning. The prompt is reparameterised through a shallow MLP with a residual connection,

$$P' = P + W_{\text{up}}\, \sigma(W_{\text{down}} P)\,, \tag{3.2}$$

and $[\, P'; E\,]$ is fed to the model. We zero-initialise W_{up} so that $P' = P$ at the start [7]. We use $m = 50$ and a bottleneck of 64. The reparameterisation network is needed only during training and can be discarded at inference.

PARA. For each encoder layer, a lightweight vector generator reads a pooled prompt representation h and emits scaling vectors that modulate the attention queries and values and the feed-forward up-projection:

$$(l_q, l_v, l_u) = \text{VG}(h), \quad Q' = l_q \odot Q,\ V' = l_v \odot V,\ U' = l_u \odot U, \tag{3.3}$$

with $\text{VG}(h) = W_{\text{up}}\, \text{GELU}(W_{\text{down}}\, \text{pool}(h)) + b_{\text{up}}$, initialised ($W_{\text{up}}$=0, b_{up}=1) so the scaling is the identity at the start [8]. As PARA targets the attention/FFN sublayers, we attach it via forward hooks on the query, value, and FFN up-projection of every encoder block

(generator bottleneck $r = 12$). PARA assumes `nn.Linear` projections; on the decoder-only DistilGPT-2 backbone, whose projections are `Conv1D`, the hook signature does not apply and PARA is reported as not run for that setting (see Chapter 6).

3.3 Proposed method: IA-DePT

Idea. DePT improves capacity/efficiency, but its short prompt P_s is static: identical for every example. In the small-model regime, where a static prompt exerts limited influence, conditioning the prompt on the input may help. IA-DePT keeps DePT's decomposition and adds a small instance gate:

$$c = W_{\text{up}}\,\text{GELU}(W_{\text{down}}\,\text{pool}(E))\,, \tag{3.4}$$

$$P'_s = P_s \odot (1 + c), \tag{3.5}$$

$$\text{input} = [\,P'_s;\; E + \Delta E\,], \tag{3.6}$$

where $\text{pool}(E)$ is a masked mean of the input word embeddings, $W_{\text{down}} \in \mathbb{R}^{g\times d}$, $W_{\text{up}} \in \mathbb{R}^{d\times g}$ (gate rank $g = 16$), and W_{up} is zero-initialised so $c = 0$ and $P'_s = P_s$ at the start. Consequently IA-DePT begins exactly as DePT and can only depart from it by learning, which gives a clean single-variable ablation (gate off = DePT, gate on = IA-DePT). The data flow is shown in Fig. 3.1.

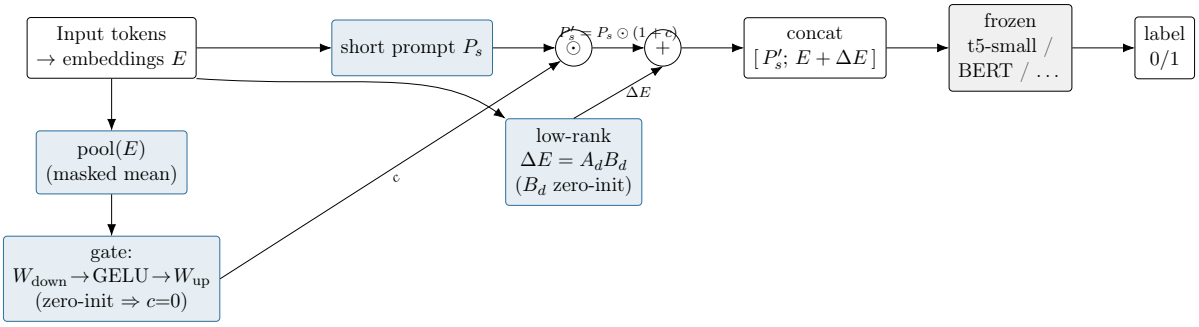


Figure 3.1: IA-DePT data flow. The DePT components (short prompt P_s and low-rank embedding update ΔE) are retained; the only addition is a zero-initialised instance gate that rescales P_s per input. At initialisation c=0, so IA-DePT ≡ DePT.

Honest positioning (novelty). Instance-conditioned prompting is not new: IDPG [10] generates an entire prompt per input. IA-DePT differs in three ways: (i) it is lightweight—a small gate over a static decomposed prompt rather than full prompt generation, giving fewer parameters and more stable optimisation on small data; (ii) it degrades exactly to DePT by construction, enabling a clean ablation; and (iii) it is evaluated head-to-head against recent (2024) methods under one controlled harness, which earlier instance-prompt work predates. The contribution is the specific lightweight combination and the controlled study, not the concept of input-dependent prompts.

Chapter 4

Experimental Setup

4.1 Backbones and tasks

The study uses five small backbones spanning three architecture families, paired with six GLUE/SuperGLUE tasks of differing size and difficulty. Table 4.1 summarises the settings. The backbones range from ≈14M (ELECTRA-small) to ≈125M (RoBERTa-base) parameters—all well below the multi-billion-parameter scale at which prompt tuning is known to rival fine-tuning, and therefore squarely in the regime this dissertation targets. RTE and MRPC are loaded from the maintained `nyu-mll/glue` Parquet mirror (identical examples to the original GLUE release); the SuperGLUE tasks WSC, CB, COPA, and WiC are loaded from the `aps/super_glue` mirror. Using maintained mirrors keeps the data pipeline stable across `datasets` versions without altering any examples.

Table 4.1: The six backbone/task settings in the unified benchmark. Sizes are total backbone parameters; "Labels" is the number of output classes; "Eval" lists the primary and secondary metrics. RTE is cast text-to-text on the encoder–decoder backbone; all other settings use a linear classification head on a pooled representation.

Task	Backbone	Architecture	Size	Labels	Eval (primary / 2nd)	Train/Val
RTE	t5-small	encoder–decoder	60.5M	2	EM-acc	2,490 / 277
WSC	DistilGPT-2	decoder-only	81.9M	2	acc / macro-F1	554 / 104
CB	ELECTRA-small	encoder-only	13.5M	3	macro-F1 / acc	250 / 56
COPA	RoBERTa-base	encoder-only	124.6M	2	acc / macro-F1	400 / 100
WiC	ELECTRA-small	encoder-only	13.5M	2	acc / MCC	5,428 / 638
MRPC	BERT-base	encoder-only	109.5M	2	F1 / acc	3,668 / 408

4.2 Training and evaluation

The backbone is frozen; only each method's module (and, for the classification settings, a small linear head) is trained. Optimisation uses AdamW; method-specific learning

rates follow the respective papers (a larger rate for prompts, a smaller rate for low-rank updates). The encoder–decoder RTE setting is trained for 2 epochs (text-to-text, exact-match accuracy via greedy generation with `num_beams=1`, `do_sample=False`, at most two new tokens); the five classification settings are trained for 3 epochs. Batch size is 16 throughout. Trainable parameter counts are reported for every method. For the headline comparison that the contribution rests on (PT vs. DePT vs. IA-DePT) we additionally report mean±std over seeds $\{0, 1, 2\}$ on RTE. The per-method configuration is summarised in Table 4.2.

Table 4.2: Per-method configuration in the unified harness. All methods share the AdamW optimiser and batch size 16; the RTE setting uses 2 epochs and the classification settings use 3. Only the trainable module and its learning rate(s) differ.

Method	**Key configuration**	**Learning rate(s)**
Fine-tuning	all backbone parameters trainable	3×10^{-4}
Prompt Tuning	$m = 50$ virtual tokens (PEFT)	1×10^{-2}
P-Tuning v2	deep prompt, $m = 20$ (PEFT prefix)	1×10^{-2}
LoPT / DPT	$m = 100$, rank $r = 10$	1×10^{-2}
DePT	prompt len. 20, rank 8	prompt 1×10^{-2} / low-rank 5×10^{-4}
ACCEPT	SCPP m=20, K=16, r=20; SCAP K=2, r=8	SCPP 1×10^{-2} / SCAP 5×10^{-4}
Residual PT	$m = 50$, bottleneck 64	1×10^{-2}
PARA	generator rank $r = 12$	1×10^{-3}
IA-DePT (ours)	prompt 20, rank 8, gate rank 16	prompt 1×10^{-2} / low-rank 5×10^{-4} / gate 1×10^{-2}

4.3 Implementation and environment

All experiments run in Google Colab on a single GPU using PyTorch with HuggingFace Transformers, Datasets, and PEFT. Each method is implemented as a small module wrapping the frozen backbone and exposing a `forward` and (for RTE) a `generate` consistent with the shared evaluation harness. To keep a free-tier GPU within memory, each method is trained, evaluated, and freed before the next, and each method is wrapped in `try/except` so one failing method does not abort the run. Two Colab-specific robustness fixes are included: the dataset is kept in native format and tensorised in a custom collate function (avoiding a `torchvision` probe that crashes some Colab images), and the GLUE/SuperGLUE mirrors are used for the data as noted above.

Chapter 5

Results on RTE (Single-Task Deep Dive)

5.1 Main comparison

Table 5.1 reports accuracy and trainable-parameter counts for all methods on RTE (t5-small) for a single seed, sorted by accuracy. Figure 5.1 visualises the accuracies, and Fig. 5.2 plots accuracy against the trainable budget (log scale), which is the trade-off frontier that RQ3 concerns.

Table 5.1: Prompt-tuning methods on GLUE/SuperGLUE RTE with t5-small (single seed). "% of model" is the trainable count relative to the full t5-small parameter count.

Method	Family	RTE acc.	# trainable	% of model
Fine-tuning	Full fine-tuning	0.6787	60,492,288	99.976
IA-DePT (ours)	**Decomposition + gate**	**0.5560**	**32,272**	**0.053**
ACCEPT	Codebook (PQ)	0.5523	22,784	0.038
LoPT	Low-rank prompt	0.5415	6,120	0.010
DPT	Low-rank prompt	0.5162	6,120	0.010
P-Tuning v2	Deep prompt	0.5054	122,880	0.203
PARA	Hidden-state adjustment	0.4801	276,552	0.457
Residual PT	Reparameterisation	0.4693	91,712	0.152
DePT	Decomposition	0.4621	15,360	0.025
Prompt Tuning	Prompt tuning	0.4513	51,200	0.085

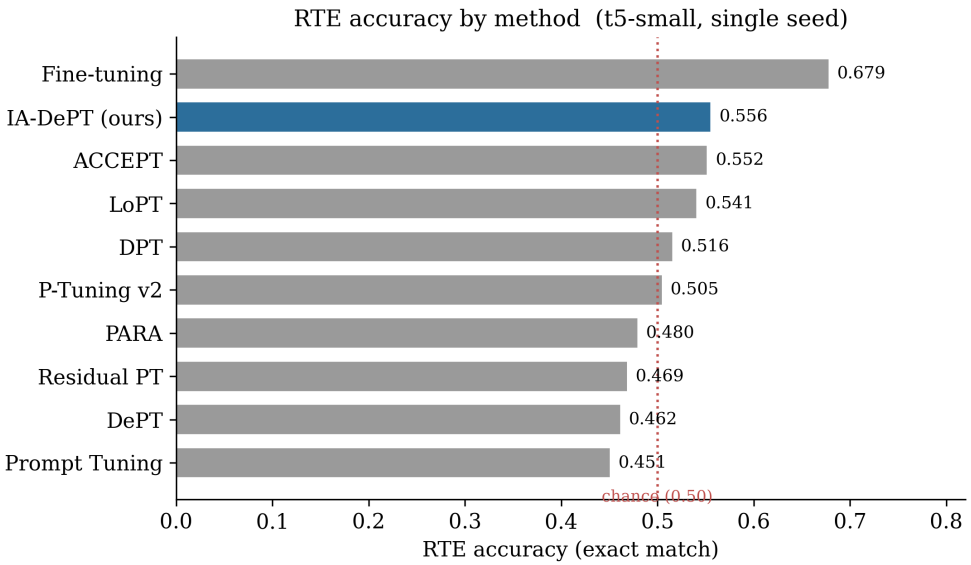


Figure 5.1: RTE accuracy by method on t5-small (single seed). The dotted line marks the 0.50 chance level for this binary task; IA-DePT (highlighted) is the strongest parameter-efficient method.

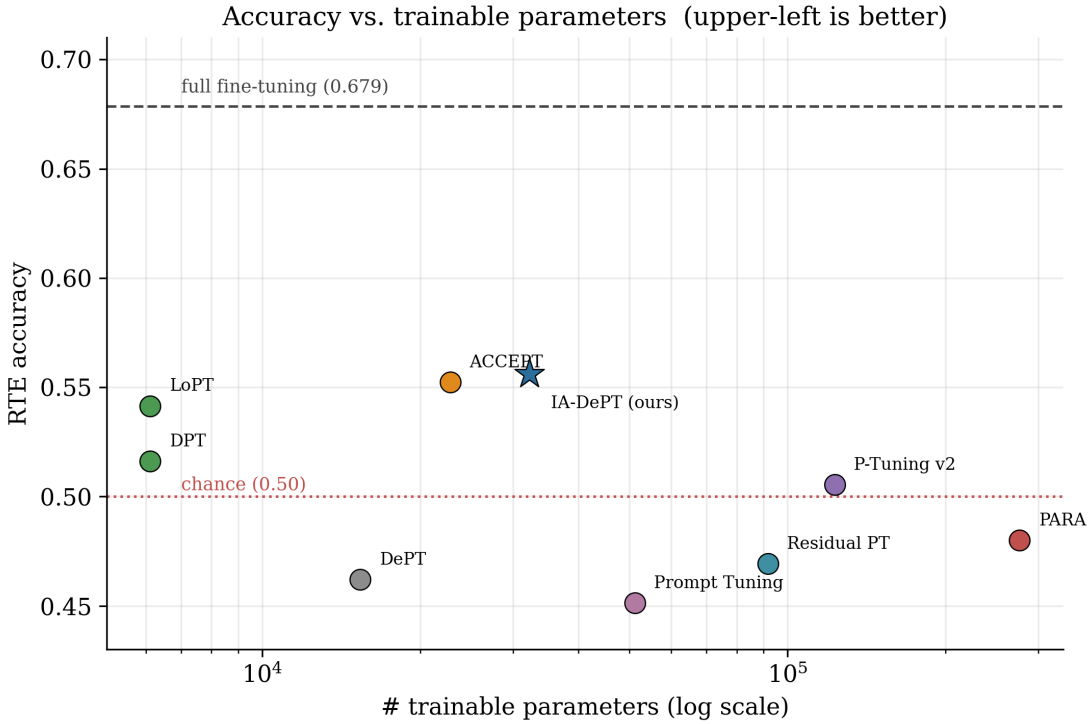


Figure 5.2: Accuracy vs. trainable parameters (log scale) for the parameter-efficient methods on RTE. Dashed line: full fine-tuning (0.679); dotted line: chance (0.50). IA-DePT and ACCEPT are the only methods clearly above chance, and they achieve this with modest ($\sim 10^4$) budgets.

5.2 Ablation: the instance gate

We isolate the contribution of instance-conditioning by comparing DePT (gate off) with IA-DePT (gate on) under otherwise identical settings, over seeds $\{0, 1, 2\}$. Table 5.2 and Fig. 5.3 report the result: the gate improves mean accuracy by 6.5 points (47.1% $\rightarrow$ 53.6%), and the improvement is larger than the combined standard deviations, so it is consistent across the three seeds rather than a single lucky run. The single-seed value in Table 5.1 (55.6% for IA-DePT, 46.2% for DePT) is the seed$=0$ point and lies within these ranges; the small difference between the single-seed and mean-over-seeds numbers is the expected effect of seed averaging.

Table 5.2: Gate ablation on RTE, mean$\pm$std over seeds $\{0, 1, 2\}$. Gate off recovers DePT exactly by construction.

Configuration	RTE acc. (mean$\pm$std)	# trainable
DePT (gate off)	0.4705 ± 0.0074	15,360
IA-DePT (gate on)	$\mathbf{0.5355 \pm 0.0196}$	32,272

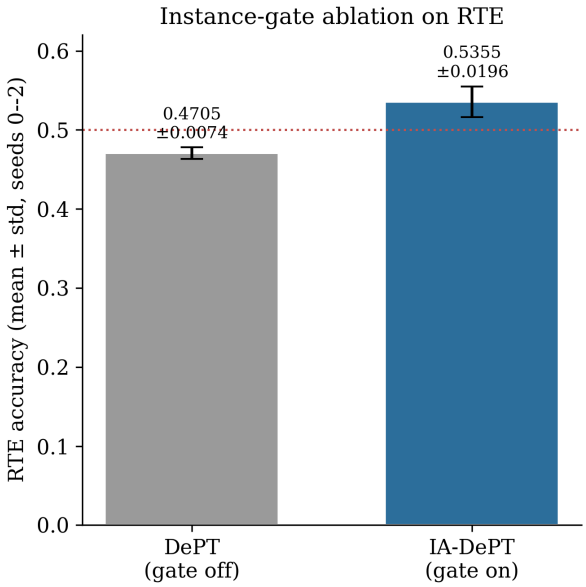


Figure 5.3: Gate ablation (mean$\pm$std over seeds 0–2). Adding the zero-initialised instance gate lifts accuracy above the chance line, at the cost of $\sim$16.9k extra parameters.

5.3 Discussion

Accuracy vs. parameters (RQ1, RQ3). Two findings stand out in Fig. 5.2. First, accuracy is not monotone in the trainable budget: the two largest PEFT modules (PARA at 277k parameters and Residual PT at 92k) sit at or below the chance line, while IA-DePT and ACCEPT reach the top of the PEFT field with roughly $2\text{–}3 \times 10^4$ parameters.

Capacity alone does not buy accuracy at this scale; where the parameters are placed matters more. Second, the two best methods share a structural feature—they augment the frozen word embeddings (DePT's ΔE, ACCEPT's SCAP) rather than relying solely on prepended tokens—suggesting that, for a small encoder, nudging the embedding space is a more parameter-efficient lever than lengthening the prompt. The two low-rank-prompt methods (LoPT and DPT) land close together as expected, since they share a mechanism, which is itself a useful internal consistency check on the harness.

Does instance-conditioning help small models? (RQ2). On RTE the answer is yes: the gate moves DePT from below the chance line to clearly above it, and does so consistently across seeds. This is a mild but encouraging counterpoint to the scale-dependency narrative—a static prompt is a weak lever on a 60M model, and letting the prompt vary with the input recovers some of the influence that a larger model would have from a static prompt alone. The effect is modest in absolute terms, which is consistent with the binding constraint at this scale being the backbone's capacity rather than the prompt's flexibility. Chapter 6 tests whether this single-task observation holds more broadly.

Threats to validity. Several caveats bound the generality of these RTE numbers, and we state them plainly. (i) *Short, equal training budget.* Every method is trained for only two epochs with lightly tuned hyperparameters. This is deliberate—it is what makes the comparison fair—but it also means the absolute accuracies are equal-budget numbers, not each method's best achievable result; the original papers train substantially longer (e.g. PARA for up to ten epochs at a smaller learning rate, ACCEPT with a learning-rate grid search). Methods that converge slowly are therefore penalised here, which partly explains why several sit near chance. (ii) *Small validation set.* RTE's 277-example development set makes small accuracy differences noisy; the multi-seed ablation mitigates but does not eliminate this. (iii) *No formal significance test.* We report mean±std and note that the gate's improvement exceeds the combined deviation, but a paired significance test (left to future work) would put the headline claim on firmer ground. None of these caveats undermines the central, honestly-scoped finding: under a fair equal-budget comparison on a small LM, a lightweight instance gate improves a strong decomposed-prompt baseline.

Training dynamics (optional). If per-epoch loss and accuracy are logged, the curves can be added here to illustrate convergence behaviour and support the equal-budget discussion above.

Chapter 6

Generalisation Across Backbones and Tasks

The RTE study fixes one backbone and one task. To answer RQ4—whether the conclusions, and the instance gate in particular, hold more broadly—we run the same method suite on the five additional settings of Table 4.1, spanning decoder-only (DistilGPT-2) and encoder-only (BERT-base, RoBERTa-base, ELECTRA-small) architectures. Figure 6.1 gives an at-a-glance view of every method on every setting; the per-setting tables follow.

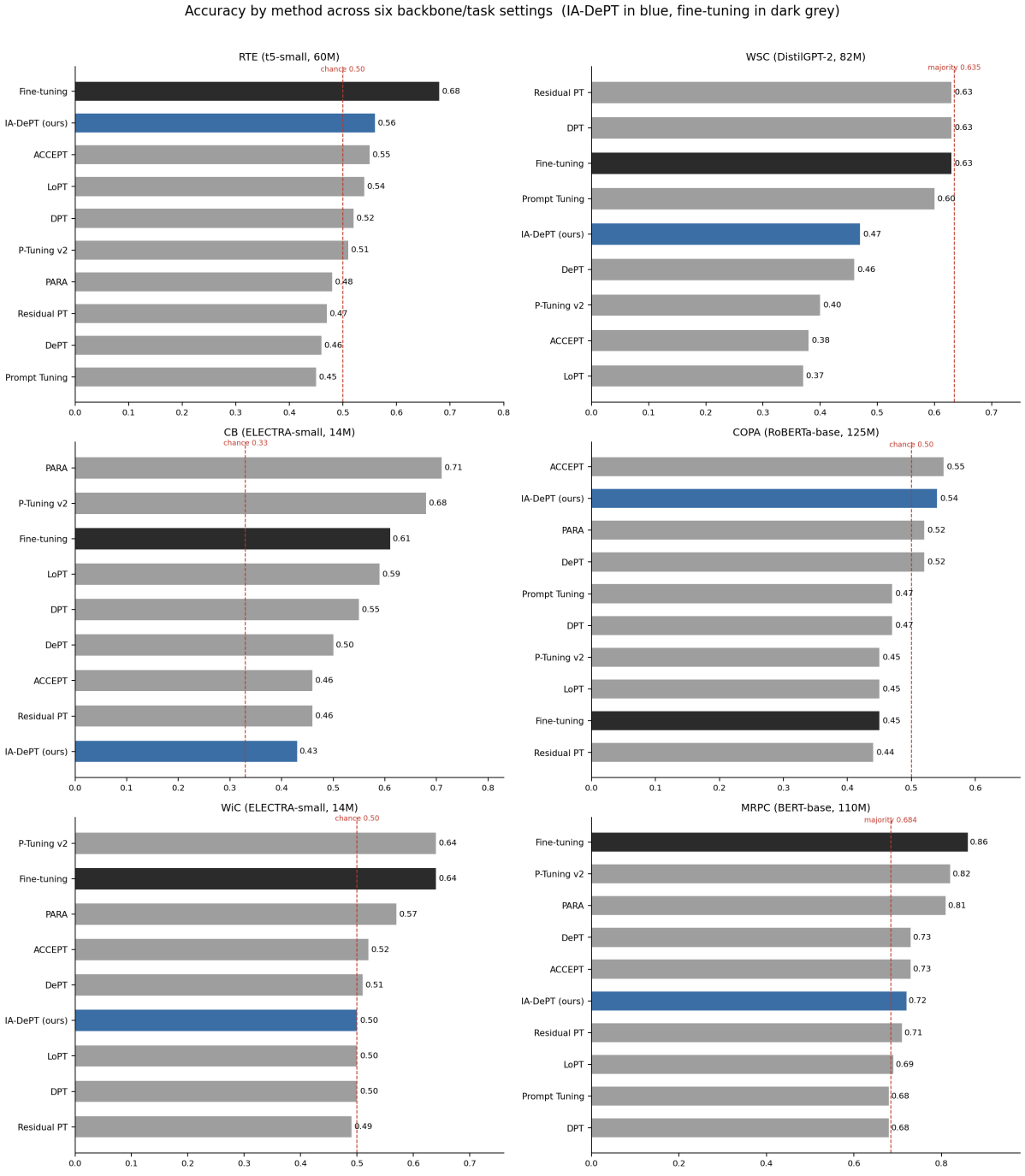


Figure 6.1: Accuracy by method across the six backbone/task settings (IA-DePT in blue, full fine-tuning in dark grey). Dotted lines mark the chance or majority-class baseline for each task. Task difficulty dominates: on the well-behaved MRPC every method clears the baseline comfortably, whereas on WiC and CB most parameter-efficient methods cluster around chance.

6.1 Per-setting results

Table 6.1: SuperGLUE WSC with DistilGPT-2 (single seed). PARA is not run on this backbone: its hooks assume `nn.Linear` projections, whereas GPT-2 uses `Conv1D`. WSC is class-imbalanced (majority-class accuracy ≈0.635), so several methods that collapse to the majority class report high accuracy but low macro-F1.

Method	Family	acc	macro-F1	# trainable	% model
Fine-tuning	Full fine-tuning	0.6346	0.4997	81,914,112	100.00
DPT	Low-rank prompt	0.6346	0.3882	10,216	0.013
Residual PT	Reparameterisation	0.6346	0.4119	139,072	0.170
Prompt Tuning	Prompt tuning	0.5962	0.4615	39,936	0.049
IA-DePT (ours)	**Decomposition + gate**	**0.4712**	**0.4699**	**49,424**	**0.060**
DePT	Decomposition	0.4615	0.4597	24,064	0.029
P-Tuning v2	Deep prompt	0.4038	0.3415	185,856	0.227
ACCEPT	Codebook (PQ)	0.3846	0.3110	31,488	0.038
LoPT	Low-rank prompt	0.3654	0.2790	10,216	0.013

Table 6.2: SuperGLUE CB with ELECTRA-small (single seed, macro-F1 primary). CB is a 3-class task with only 250 training and 56 validation examples. Prompt Tuning is not reported: under the longer CB sequence length, the PEFT prompt-tuning configuration raised a sequence-length mismatch.

Method	Family	macro-F1	acc	# trainable	% model
PARA	Hidden-state adjustment	0.4986	0.7143	277,395	2.047
P-Tuning v2	Deep prompt	0.4684	0.6786	59,651	0.440
Fine-tuning	Full fine-tuning	0.3874	0.6071	13,549,571	100.00
DPT	Low-rank prompt	0.3789	0.5536	3,051	0.023
LoPT	Low-rank prompt	0.3756	0.5893	3,051	0.023
Residual PT	Reparameterisation	0.2898	0.4643	23,747	0.175
ACCEPT	Codebook (PQ)	0.2663	0.4643	14,851	0.110
IA-DePT (ours)	**Decomposition + gate**	**0.2372**	**0.4286**	**10,643**	**0.079**
DePT	Decomposition	0.2222	0.5000	6,403	0.047

Table 6.3: SuperGLUE COPA with RoBERTa-base (single seed, accuracy primary). COPA is a hard binary causal-reasoning task with 400 training and 100 validation examples; full fine-tuning collapses to near chance here, while two embedding-augmenting PEFT methods sit at the top.

Method	Family	acc	macro-F1	# trainable	% model
ACCEPT	Codebook (PQ)	0.5500	0.3548	31,490	0.025
IA-DePT (ours)	**Decomposition + gate**	**0.5400**	**0.3506**	**49,426**	**0.040**
DePT	Decomposition	0.5200	0.4286	24,066	0.019
PARA	Hidden-state adjustment	0.5200	0.5192	831,122	0.667
DPT	Low-rank prompt	0.4700	0.3629	10,218	0.008
Prompt Tuning	Prompt tuning	0.4700	0.3960	630,530	0.506
Fine-tuning	Full fine-tuning	0.4500	0.3103	124,647,170	100.00
LoPT	Low-rank prompt	0.4500	0.3103	10,218	0.008
P-Tuning v2	Deep prompt	0.4500	0.3103	185,858	0.149
Residual PT	Reparameterisation	0.4400	0.3994	139,074	0.112

Table 6.4: SuperGLUE WiC with ELECTRA-small (single seed, accuracy primary; MCC secondary). WiC is a notoriously hard word-sense task (chance 0.50); most parameter-efficient methods sit at or near chance, and the deep-prompt and full-fine-tuning baselines lead. Prompt Tuning is not reported for the same configuration reason noted for CB.

Method	Family	acc	MCC	# trainable	% model
P-Tuning v2	Deep prompt	0.6411	0.2840	59,394	0.438
Fine-tuning	Full fine-tuning	0.6395	0.2951	13,549,314	100.00
PARA	Hidden-state adjustment	0.5705	0.2301	277,138	2.045
ACCEPT	Codebook (PQ)	0.5172	0.0429	12,546	0.093
DePT	Decomposition	0.5110	0.0681	5,122	0.038
DPT	Low-rank prompt	0.5000	0.0000	2,794	0.021
LoPT	Low-rank prompt	0.5000	0.0000	2,794	0.021
IA-DePT (ours)	**Decomposition + gate**	**0.5000**	**0.0000**	**9,362**	**0.069**
Residual PT	Reparameterisation	0.4906	-0.0371	23,490	0.173

Table 6.5: GLUE MRPC with BERT-base (single seed, F1 primary). MRPC is the best-behaved task in the suite: full fine-tuning reaches 0.90 F1 and every parameter-efficient method clears 0.79 F1.

Method	Family	F1	acc	# trainable	% model
Fine-tuning	Full fine-tuning	0.9002	0.8603	109,483,778	100.00
PARA	Hidden-state adjustment	0.8764	0.8113	831,122	0.759
P-Tuning v2	Deep prompt	0.8624	0.8162	185,858	0.170
IA-DePT (ours)	**Decomposition + gate**	**0.8263**	**0.7157**	**49,426**	**0.045**
DePT	Decomposition	0.8237	0.7304	24,066	0.022
Residual PT	Reparameterisation	0.8187	0.7059	139,074	0.127
Prompt Tuning	Prompt tuning	0.8122	0.6838	39,938	0.037
DPT	Low-rank prompt	0.8122	0.6838	10,218	0.009
ACCEPT	Codebook (PQ)	0.8062	0.7255	31,490	0.029
LoPT	Low-rank prompt	0.7921	0.6887	10,218	0.009

6.2 Does the instance gate generalise? (RQ4)

The clean ablation built into IA-DePT (gate off = DePT) lets us read the effect of instance-conditioning directly off each setting. Table 6.6 and Fig. 6.2 collect the comparison on each setting's primary metric.

Table 6.6: Effect of the instance gate (IA-DePT vs. DePT) on each setting's primary metric. RTE is the three-seed mean; the others are single seed. Δ is IA-DePT − DePT. Positive Δ indicates the gate helps.

Setting	Primary metric	DePT	IA-DePT	Δ
RTE (t5-small)	acc (3-seed)	0.4705	0.5355	+0.0650
WSC (DistilGPT-2)	acc	0.4615	0.4712	+0.0097
CB (ELECTRA-small)	macro-F1	0.2222	0.2372	+0.0150
COPA (RoBERTa-base)	acc	0.5200	0.5400	+0.0200
WiC (ELECTRA-small)	acc	0.5110	0.5000	−0.0110
MRPC (BERT-base)	F1	0.8237	0.8263	+0.0026

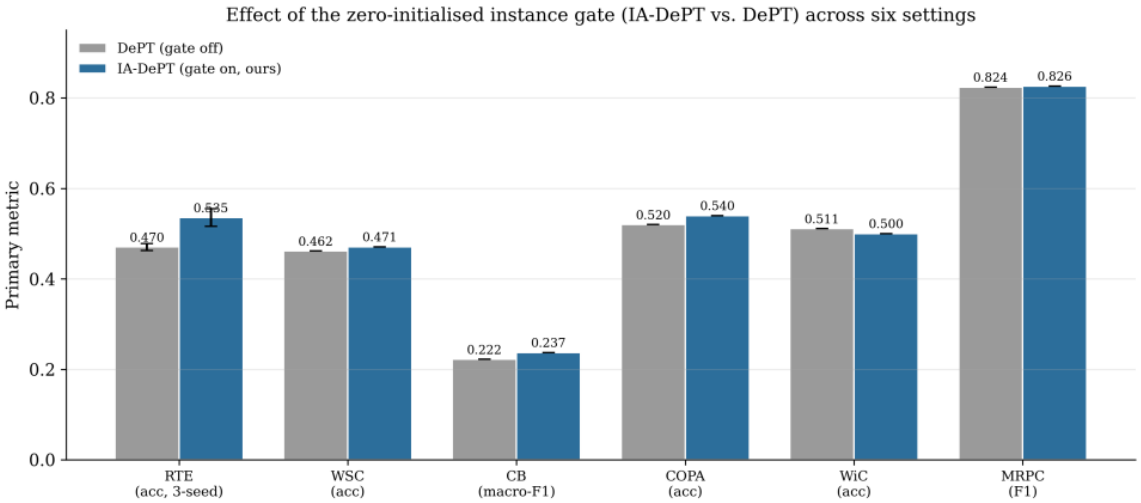


Figure 6.2: Effect of the zero-initialised instance gate (IA-DePT vs. DePT) across the six settings, on each setting's primary metric. RTE shows the three-seed mean with std bars. The gate helps in five of six settings; the only regression (WiC) is small and occurs where every PEFT method sits at chance.

6.3 Discussion

The gate helps broadly, but not universally. On the primary metric, IA-DePT improves on DePT in five of the six settings (Table 6.6), with the largest and most reliable gain on RTE (+6.5 points, three-seed). The gains on WSC, COPA, CB, and MRPC are smaller but consistent in sign, and on WSC the gate also raises macro-F1 (0.4597 $\rightarrow$ 0.4699), indicating it does not simply trade calibration for headline accuracy. The single regression is on WiC, where IA-DePT, DePT, and the low-rank-prompt methods all collapse to the 0.50 chance accuracy (MCC $\approx$0): when no method extracts signal, the gate has nothing to amplify, and the tiny difference is within noise. The honest summary is therefore that input-conditioning a short decomposed prompt is a broadly beneficial, low-cost modification for small models, rather than a guaranteed improvement on every task.

Task difficulty dominates the small-model regime. The clearest cross-setting pattern in Fig. 6.1 is that the task matters more than the method. On MRPC—a large, well-behaved paraphrase task—every method, including the cheapest low-rank prompts, clears 0.79 F1 and full fine-tuning reaches 0.90. On WiC and CB, by contrast, most parameter-efficient methods hover around chance regardless of their parameter budget, and even full fine-tuning is only modestly above baseline. This reinforces the RTE finding that, at small scale, the binding constraint is how much task signal the frozen backbone

plus a tiny module can extract, not the size of the module.

Where the parameters are placed still matters more than how many. The non-monotonicity observed on RTE recurs across settings. On COPA, the 124M-parameter full fine-tune lands at chance (0.45) while IA-DePT and ACCEPT—both <50k trainable parameters and both augmenting the embeddings—are the two best methods. Deep-prompt P-Tuning v2 and hidden-state PARA, the two largest PEFT modules, are strong on the encoder-only ELECTRA settings (WiC, CB) but middling elsewhere, and PARA cannot be applied to the decoder-only GPT-2 backbone without re-engineering its hooks. No single family dominates across architectures; the embedding-augmenting decomposition and codebook methods are the most consistent lightweight performers.

Threats to validity (cross-setting). The RTE caveats carry over and are compounded here. (i) *Single seed for the new settings.* Only RTE has a multi-seed estimate; the five additional settings are single-seed, and the validation sets are small (WSC 104, CB 56, COPA 100), so individual numbers are noisy and small Δ values (e.g. MRPC +0.0026) should be read as "no regression" rather than a confident gain. (ii) *Equal, short budget.* As on RTE, every method gets the same short budget, which penalises slow-converging methods and is not their best achievable result. (iii) *Two methods are missing from two settings* (PARA on WSC; Prompt Tuning on CB and WiC) for the architecture/configuration reasons stated in the table captions; their absence is reported rather than silently imputed. These do not change the qualitative conclusions but do mean the cross-setting Δ values are best treated as directional evidence that the RTE result is not an isolated artifact.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This dissertation studied prompt tuning in the small-language-model regime through a controlled benchmark and a lightweight new method. Re-implementing the major prompt-tuning families in one harness and deep-diving on t5-small/RTE (RQ1, RQ3) showed that accuracy is not driven by the size of the trainable module: the two strongest parameter-efficient methods, IA-DePT and ACCEPT, reach the top of the field with $\sim 10^4$ parameters, while larger modules sit near chance, and both top methods share the trait of augmenting the frozen embeddings rather than only prepending tokens. On the instance-conditioning question (RQ2), the proposed IA-DePT—DePT plus a zero-initialised instance gate—improved over DePT by 6.5 points (53.6% vs. 47.1%, mean over three seeds) for the cost of ∼16.9k extra parameters, and was the best PEFT method overall on a single seed. Because the gate degrades exactly to DePT at initialisation, this is a clean single-variable result.

Extending the benchmark to six backbone/task settings spanning encoder–decoder, encoder-only, and decoder-only architectures (RQ4) showed that the instance gate improves on DePT in five of the six settings on each setting's primary metric, regressing only marginally on WiC, where every PEFT method sits at chance. The cross-architecture study also surfaced a clearer signal than any single method result: in the small-model regime, task difficulty and the *placement* of trainable parameters (embedding-augmenting vs. prompt-prepending) matter more than the raw parameter budget. The absolute accuracies remain modest on the harder tasks, which is itself consistent with the scale-dependency literature and is discussed honestly above.

7.2 Future work

- **More seeds and significance.** Extend the multi-seed protocol from RTE to all six settings and add a paired significance test for the IA-DePT-vs-DePT comparison, putting the cross-setting Δ values on firmer statistical ground.
- **Longer / tuned budgets.** Re-run the strongest methods at their papers' training budgets to separate "slow to converge" from "genuinely weak at small scale", complementing the equal-budget comparison.
- **Additional baselines.** Include IDPG—the closest prior art to IA-DePT—and further recent instance-aware methods for a more complete comparison.
- **Deep instance-conditioning.** Extend IA-DePT so the instance signal also modulates hidden states at each layer (reusing PARA-style hooks where the architecture allows), moving from an input-layer gate toward deep instance-conditioned prompts, and adapt the hooks to `Conv1D` projections so the approach also covers GPT-2-style decoders.
- **Broader coverage.** Add a second backbone at the upper edge of the small regime (e.g. t5-base) to probe directly how the conclusions shift as scale grows.

Bibliography

[1] B. Lester, R. Al-Rfou, and N. Constant. The Power of Scale for Parameter-Efficient Prompt Tuning. *EMNLP*, 2021.

[2] X. L. Li and P. Liang. Prefix-Tuning: Optimizing Continuous Prompts for Generation. *ACL*, 2021.

[3] X. Liu et al. P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks. *ACL*, 2022.

[4] Y. Xiao, L. Xu, J. Li, W. Lu, and X. Li. Decomposed Prompt Tuning via Low-Rank Reparameterization. *Findings of EMNLP*, 2023.

[5] Z. Shi and A. Lipani. DePT: Decomposed Prompt Tuning for Parameter-Efficient Fine-tuning. *ICLR*, 2024.

[6] Y.-C. Lin, W.-H. Li, J.-C. Chen, and C.-S. Chen. ACCEPT: Adaptive Codebook for Composite and Efficient Prompt Tuning. *arXiv:2410.12847*, 2024.

[7] A. Razdaibiedina, Y. Mao, M. Khabsa, M. Lewis, R. Hou, J. Ba, and A. Almahairi. Residual Prompt Tuning: Improving Prompt Tuning with Residual Reparameterization. *Findings of ACL*, 2023.

[8] Z. Liu, Y. Zhao, M. Tan, W. Zhu, and A. X. Tian. PARA: Parameter-Efficient Fine-tuning with Prompt Aware Representation Adjustment. *EMNLP (Industry Track)*, 2024.

[9] S. Guo, S. Damani, and K.-H. Chang. LoPT: Low-Rank Prompt Tuning for Parameter-Efficient Language Models. *arXiv:2406.19486*, 2024.

[10] Z. Wu, S. Wang, J. Gu, R. Hou, Y. Dong, et al. IDPG: An Instance-Dependent Prompt Generation Method. *NAACL*, 2022.

[11] Z. Li and N. Collier. A Survey on Prompt Tuning. *arXiv:2507.06085*, 2025.

[12] V. Lialin, V. Deshpande, and A. Rumshisky. Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning. *arXiv:2303.15647*, 2023.

[13] T. Vu, B. Lester, N. Constant, R. Al-Rfou, and D. Cer. SPoT: Better Frozen Model Adaptation through Soft Prompt Transfer. *ACL*, 2022.

[14] A. Asai, M. Salehi, M. E. Peters, and H. Hajishirzi. ATTEMPT: Parameter-Efficient Multi-task Tuning via Attentional Mixtures of Soft Prompts. *EMNLP*, 2022.

[15] Z. Wang, R. Panda, L. Karlinsky, R. Feris, H. Sun, and Y. Kim. Multitask Prompt Tuning Enables Parameter-Efficient Transfer Learning. *ICLR*, 2023.

[16] M. Wu et al. Parameter-Efficient Multi-task Fine-tuning by Learning to Transfer Token-wise Prompts. *Findings of EMNLP*, 2023.

[17] E. J. Hu et al. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv:2106.09685*, 2021.

[18] N. Houlsby et al. Parameter-Efficient Transfer Learning for NLP. *ICML*, 2019.

[19] E. Ben-Zaken, S. Ravfogel, and Y. Goldberg. BitFit: Simple Parameter-Efficient Fine-tuning for Transformer-based Masked Language Models. *arXiv:2106.10199*, 2021.

[20] C. Raffel et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *JMLR*, 2020.

[21] A. Wang et al. SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems. *NeurIPS*, 2019.

[22] A. Wang et al. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *ICLR*, 2019.

[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *NAACL*, 2019.

[24] Y. Liu et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692*, 2019.

[25] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. *ICLR*, 2020.

[26] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv:1910.01108*, 2019.

[27] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language Models are Unsupervised Multitask Learners. Technical report, OpenAI, 2019.

[28] W. B. Dolan and C. Brockett. Automatically Constructing a Corpus of Sentential Paraphrases. *IWP*, 2005.