Propulsion: Steering LLM with Tiny Fine-Tuning

Md Kowsher[1], Nusrat Jahan Prottasha[1], Prakash Bhat[2,‡]
[1]University of Central Florida, FL, USA, [2]Amazon, USA
{md.kowsher, nusrat.prottasha}@ucf.edu, bhatprak@amazon.com

Abstract

The rapid advancements in Large Language Models (LLMs) have revolutionized natural language processing (NLP) and adjacent fields, yet fine-tuning these models for specific tasks remains computationally expensive and risks degrading pre-learned features. To address these challenges, we propose *Propulsion*, a novel parameter-efficient fine-tuning (PEFT) method designed to optimize task-specific performance while drastically reducing computational overhead. Inspired by the concept of controlled adjustments in physical motion, *Propulsion* selectively re-scales specific dimensions of a pre-trained model, guiding output predictions toward task objectives without modifying the model's parameters. By introducing lightweight, trainable *Propulsion* parameters at the pre-trained layer, we minimize the number of parameters updated during fine-tuning, thus preventing the overfitting or overwriting of existing knowledge. Our theoretical analysis, supported by Neural Tangent Kernel (NTK) theory, shows that *Propulsion* approximates the performance of full fine-tuning with far fewer trainable parameters. Empirically, *Propulsion* reduces the parameter count from 355.3 million to a mere 0.086 million—achieving over a 10x reduction compared to standard approaches like LoRA—while maintaining competitive performance across benchmarks.

1 Introduction

Training large language models consumes significant computational resources, sometimes taking up to six months (Zhao et al., 2023). This creates bottlenecks in AI development and raises environmental concerns (Rillig et al., 2023). To mitigate this, we often fine-tune pre-trained models like BERT (Devlin et al., 2018), GPT (Mann et al., 2020), and RoBERTa (Liu et al., 2019) instead of training from scratch. However, fine-tuning these pre-trained models is still challenging due to their large sizes; for instance, modern LLMs can have up to 7 billion parameters (Jiang et al., 2023; Touvron et al., 2023; Almazrouei et al., 2023; Le Scao et al., 2023). Traditional full model fine-tuning is effective but often too expensive and inefficient, limited by computational resources and time (Bender et al., 2021; Kim et al., 2024; Wu et al., 2024).

Recent advances have explored the realm of PEFT (Xu et al., 2023; Kowsher et al., 2023) techniques as a solution to these challenges. Methods such as adapter layers (Lin et al., 2020; Houlsby et al., 2019), prompt tuning (Lester et al., 2021), low-rank adaptation (Hu et al., 2021), quantization (Gray and Neuhoff, 1998), selective row or columns tuning (Kowsher et al., 2024), and lightweight fine-tuning (Liu et al., 2021a) alternatives propose modifications that require adjusting only a fraction of the model's total parameters. These approaches, while promising, often involve trade-offs between efficiency, performance, and adaptability, thus there is still room to improve the combined utility. To address the limitations of existing PEFT methods, we introduce *Propulsion*: a novel approach for fine-tuning that leverages the observation that small, targeted changes in the output vectors of a model's layers can lead to substantial shifts in the model's overall behavior. In physical dynamics, propulsion can steer or change an object's trajectory through small, controlled bursts of force (Turchi, 1998; Budashko, 2020). Similarly, our *Propulsion* method applies minimal yet strategic adjustments or re-scaling to the pre-trained dimensions of a neural network, as effectively "steering" the model's responses towards desired outcomes with minimal energy expenditure and maximal retention of pre-learned features. To do this, we introduce a series of trainable linear parameters—denoted as "*Propulsion* parameters". These parameters are finely tuned to amplify or attenuate

[‡]This work does not relate to Prakash's position at Amazon.

Proceedings of the 31st International Conference on Computational Linguistics, pages 7569–7597
January 19–24, 2025.

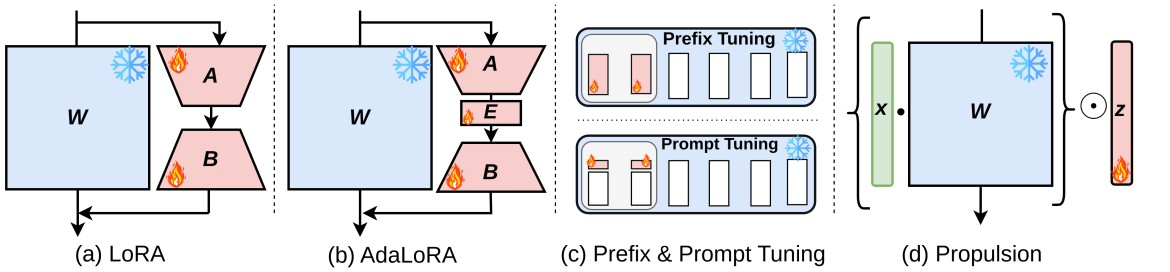


Figure 1: A detailed illustration of the model architectures for five different adapters: (a) LoRA, (b) AdaLoRA, (c) Prefix & Prompt Tuning, and (d) Propulsion. In the diagrams, $\boldsymbol{W}$ represents the pre-trained weight matrix, which is kept frozen, while $\boldsymbol{X}$ denotes the input. The matrices $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{E}$ are trainable and of lower rank. The variable $\mathbf{z}$ indicates the *Propulsion* parameter.

specific aspects of the model's behavior, thereby optimizing performance on specific tasks with minimal computational overhead. Figure 1 compares the different PEFT methods with our *Propulsion* approach.

To support our method theoretically, we analyze *Propulsion* in the context of the NTK framework. The NTK, introduced by Jacot et al. (2018), characterizes the training dynamics of neural networks in the regime where the width of the network tends to infinity. Under this framework, it has been shown that fine-tuning methods such as LoRA approximate the full fine-tuning of neural networks by focusing on a low-rank subspace (Jang et al., 2024; Tomihari and Sato, 2024). Similarly, our analysis demonstrates that *Propulsion* closely approximates the NTK of full fine-tuning by updating only a diagonal subset of the model's parameters. This theoretical grounding ensures that *Propulsion* achieves similar performance to full fine-tuning, despite its significantly reduced computational requirements.

We evaluate the effectiveness of our approach across several benchmarks on different language models. Our experimental results show that *Propulsion* outperforms current PEFT techniques while requiring fewer trainable parameters. For instance, *Propulsion* uses about 12 times fewer parameters than AdaLoRA and achieves higher accuracy (details in Section 4).

2 Propulsion

We introduce a clear outline of the *Propulsion* concept and its practical benefits. Consider that we have a pre-trained language model $\mathbb{M}$ with N layers, such as $L = \{L_1, L_2, \ldots, L_N\}$, where we freeze all parameters. We represent any given input as $x \in \mathbb{R}^{s \times d_{in}}$, where s denotes the sequence length of tokens, d represents the dimension of each token, and x can be any hidden layer's output or input of next following layer of the neural networks, Key, Queries, and Values, and so on. Given x as the input, we extract $V_i = L_i(x; W) \in \mathbb{R}^{s \times d_{out}}$ with pre-trained frozen weight $W \in \mathbb{R}^{d_{in} \times d_{out}}$.

To introduce task-specific modifications, we initialize a trainable *Propulsion* matrix $\mathscr{Z} \in \mathbb{R}^{N \times d_{out}}$, where $\mathbf{z_i} = \{z_1, z_2, \ldots, z_{d_{out}}\} \in \mathscr{Z}$. Each $\mathbf{z_i}$ performs an element-wise scalar transformation to each corresponding element $\mathbf{v_j} \in V_i$ to steer the output projection of L_i, where $\mathbf{v_j} = \{v_1, v_2 \ldots v_{d_{out}}\}$ represents the $j-th$ token representation of output V_i from layer L_i.

We train $\mathbf{z_i}$ by calculating the element-wise multiplication $\mathbf{v_j} \odot \mathbf{z_i}$ to generate $\mathbf{v_j}'$, where $\odot$ denotes the element-wise multiplication operation performed between $z_{d_{out}}$ and every element $v_{d_{out}}$ within the output vector $\mathbf{v_j}$. We can define this operation as :

$$\mathbf{v_j}' = [v_1 \cdot z_1, v_2 \cdot z_2, \ldots, v_{d_{out}} \cdot z_{d_{out}}] \tag{1}$$

Similarly, by following Equation 1; for all s tokens, we can steer the output of V_i by training the *Propulsion* $\mathbf{z_i}$, which can be defined as :

$$V_i' = [\mathbf{v_1} \odot \mathbf{z_i}, \mathbf{v_2} \odot \mathbf{z_i}, \ldots, \mathbf{v_s} \odot \mathbf{z_i}] \tag{2}$$

Once V_i' has been calculated, it is used as the next input to extract the output of the next layer. So the transformed output V_i' of layer L_i is used as the input x to layer L_{i+1}

We enhance the *Propulsion* concept by incorporating polynomial scaling to the *Propulsion* parameter $\mathbf{z_i}$. By raising $\mathbf{z_i}$ to the power of k, termed as polynomial scaling, we allow for a more flexible and dynamic adjustment of the model's responses to input features. This scaling adjusts the magnitude of the propulsion effect, providing a method

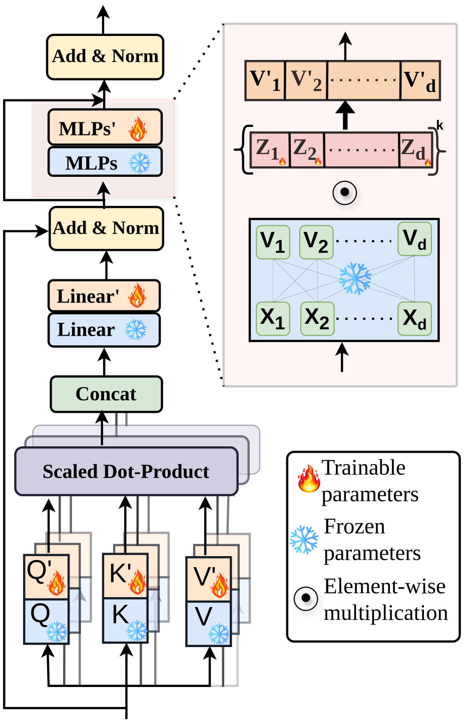


Figure 2: *Propulsion* in Transformer Block. Within the figure, the red cells represent trainable parameters while the blue cells represent the frozen parameters. The *Propulsion* layers above shows where our method executes during model fine-tuning. All layers use the same *Propulsion* matrix, but are modified by their corresponding vector z_i.

to vary the influence of the propulsion parameters across different stages of learning or different parts of the data. We can define this operation as :

$$V_i' = [\mathbf{v_1} \odot \mathbf{z_i}^k, \mathbf{v_2} \odot \mathbf{z_i}^k, \ldots, \mathbf{v_s} \odot \mathbf{z_i}^k] \qquad (3)$$

In Figure 2, we illustrate the general structure of our *Propulsion* method in the Transformer block that modifies the output of K, Q, V, and MLP matrix through element-wise multiplication with *Propulsion* trainable parameters to fine-tune the LLMs efficiently.

3 Neural Tangent Kernel (NTK) Analysis

The NTK, introduced by Jacot et al. (2018), characterizes how small changes in a network's parameters affect its output. In the NTK regime, where the width of the network becomes very large, the training dynamics of neural networks are determined by the NTK, which remains nearly constant during training (Afzal et al.).

In this section, we analyze the Propulsion method in the NTK regime and show that the NTK of Propulsion approximates the NTK of full fine-tuning.

Theorem 1 *Let $\phi_P(\mathbf{x};\theta_t)$ be the output of the Propulsion model at time step t, where the base matrix θ_0 is pre-trained and fixed, and the Propulsion matrix $\mathbf{z}_t$ is updated during training. Let $\phi_F(\mathbf{x};\theta_t)$ be the output of the fully fine-tuned model at time step t. Under the NTK regime, where the width d of the network is sufficiently large, the NTK for Propulsion fine-tuning approximates the NTK for full fine-tuning with high probability. Formally, for inputs $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$, the NTK for Propulsion satisfies:*

$$\mathbf{K}^F\left(\mathbf{x}, \mathbf{x}'\right) \approx \mathbf{K}^P\left(\theta_0 \mathbf{x}_i, \theta_0 \mathbf{x}_j\right)$$

Furthermore, the error between the NTK for Propulsion and the NTK for full fine-tuning can be bounded using the Johnson-Lindenstrauss Lemma. Specifically, for any $\varepsilon > 0$ and constant c, with high probability:

$$\Pr\left[\left|(\theta_0 \mathbf{x}_i)^\top (\theta_0 \mathbf{x}_j) - \mathbf{x_i}^\top \mathbf{x_j}\right|\right] \geq 1 - 4\exp\left(-\frac{(\varepsilon^2 - \varepsilon^3)d}{4}\right)$$

The full theoretical proof of this theorem is provided in **Appendix A**. Additionally, in **Appendix B**, we present the empirical results supporting this theory, and in **Appendix C**, we provide a detailed analysis of the NTK regime of Propulsion.

4 Experiments

We evaluate our methods on NLP tasks, including the General Language Understanding Evaluation (GLUE) benchmark, question answering, text summarization, common sense reasoning, and arithmetic reasoning. The details of the training and algorithm are described in Appendix E.

4.1 Baselines

We use well-known PEFT methods for our baseline comparisons, including Adapter (Houlsby et al., 2019), Prompt Tuning (Lester et al., 2021), Prefix-Tuning (Li and Liang, 2021), $(IA)^3$ (Liu et al., 2022a), Bitfit (Zaken et al., 2021), LoRA (Hu et al., 2021), AdaLoRA (Zhang et al., 2023), MAM Adapter (He et al., 2021), PROPETL (Zeng et al., 2023), LoKr (Edalati et al., 2022), and LoHa (Hyeon-Woo et al., 2021). The implementations used for these methods come from the Hugging Face (Mangrulkar et al., 2022). The experimental setup follows that of Xu et al. (2023) for the GLUE benchmark; for the question answering and text summarizing datasets, we have followed Zhang et al. (2023).

Model	PEFT Method	#TPs	CoLA	SST2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	Avg.
RoB_B	FT	124.6M	59.84	92.89	85.24/88.18	90.48/90.16	90.18/87.02	86.27	91.17	72.43	83.56 /88.45
	Adapter^S	7.41M	60.32	92.14	89.24/ 85.29	90.25/90.09	**90.81/86.55**	**87.33**	90.84	73.56	84.31/87.31
	Prompt tuning	0.61M	49.37	91.09	74.83/72.72	82.44/83.11	82.99/78.35	80.57	80.03	58.12	74.93/78.06
	Prefix-tuning	0.96M	55.31	92.17	87.25/83.24	88.48/88.32	87.75/84.09	85.21	90.77	54.51	80.18/85.21
	(IA)^3	0.66M	59.58	92.02	87.00/82.52	90.30/90.32	87.99/84.10	83.95	90.88	71.12	82.85/85.64
	BitFit	0.086M	61.38	92.67	88.22/84.41	90.34/90.27	88.12/84.11	84.64	91.09	75.58	84.20/86.26
	LoRA	0.89M	60.09	92.40	88.50/84.68	90.66/90.83	88.83/85.21	86.54	92.02	72.92	83.99/86.90
	AdaLoRA	1.03M	59.82	91.69	88.99/85.03	90.83/90.73	88.58/84.98	86.26	91.43	70.04	83.45/86.91
	MAM Adapter	46.78M	58.42	**93.19**	89.31 /85.21	90.74/90.42	88.31/83.20	86.63	90.19	72.62	83.67/86.27
	PROPETL $_{Adapter}$	1.87M	**63.11**	92.18	85.25/81.82	91.33 /**91.04**	89.22/85.79	86.49	92.56	75.54	84.46/86.21
	PROPETL $_{Prefix}$	10.49M	60.18	91.36	86.73/84.98	90.30/90.19	88.54/85.05	86.22	91.51	63.31	82.26/86.74
	PROPETL $_{LoRA}$	1.77M	61.72	92.54	87.42/83.87	90.76/90.55	88.90/85.55	86.84	92.04	67.39	83.45/86.65
	Propulsion(All)	0.086M	61.76	93.18	**89.34/ 85.99**	**91.37**/90.92	89.11/86.53	86.41	**92.79**	**75.66**	**84.95/87.81**
	Propulsion(Attn)	**0.028M**	58.51	92.03	89.01/85.14	89.36/89.96	86.73/84.80	85.13	89.89	75.02	83.21/86.63
RoB_L	FT	355.3M	65.78	95.50	92.22/94.28	91.74/91.96	90.83/88.68	89.21	93.19	81.40	87.48/91.64
	Adapter^S	19.77M	62.03	94.65	90.19/87.94	92.58/92.42	92.19/88.50	91.00	94.31	81.25	87.27/89.62
	Prompt-tuning	1.07M	60.22	93.61	79.04/76.29	78.51/78.99	80.74/75.16	68.15	89.13	60.29	76.21/76.81
	Prefix-tuning	2.03M	59.01	93.76	88.24/86.37	90.92/91.07	88.88/85.45	89.30	93.32	74.01	84.68/87.63
	(IA)^3	1.22M	60.17	94.61	90.52/87.33	92.22/86.25	89.45/86.25	88.63	94.25	81.23	86.38/86.61
	Bitfit	0.225M	**66.72**	95.10	90.70/88.38	91.93/**93.38**	89.48/86.43	89.98	94.47	85.73	88.01/89.39
	LoRA	1.84M	64.47	**95.67**	90.50/86.19	91.66/91.44	90.15/86.91	90.76	95.00	79.78	87.24/88.18
	AdaLoRA	2.23M	65.85	94.95	**91.46**/87.34	92.05/91.80	89.60/86.30	90.36	94.62	77.98	88.20/88.48
	MAM Adapter	122.20M	64.39	95.08	90.12/87.77	92.44/92.18	90.87/86.65	90.62	94.31	86.62	88.05/88.86
	PROPETL $_{Adapter}$	5.40M	65.55	94.82	89.71/86.54	91.92/91.67	90.67/87.74	**91.37**	95.20	**85.89**	88.14/88.65
	PROPETL $_{Prefix}$	26.85M	62.24	94.17	90.04/87.92	90.70/90.49	89.30/86.30	90.33	94.73	79.71	86.40/88.23
	PROPETL $_{LoRA}$	4.19M	61.90	94.93	89.06/86.19	91.66/91.38	90.93/88.05	90.53	94.93	82.57	87.06/88.54
	Propulsion(All)	0.225M	64.53	95.10	90.47/**88.85**	**92.78**/92.58	**92.26/88.91**	90.52	**95.34**	85.30	**88.28/90.11**
	Propulsion(Attn)	**0.073M**	62.31	94.02	89.78/87.95	90.16/90.86	88.02/86.19	89.54	94.00	83.07	86.36/88.33

Table 1: Performance Comparison of RoBERTa Models on GLUE Tasks: Metrics include MCC for CoLA, Accuracy for SST-2, Accuracy/F1-score for MRPC and QQP, Pearson/Spearman correlation for STS-B, and Accuracy for MNLI, QNLI, and RTE. "Propulsion(All)" applies Propulsion to all layers (Embedding, MLP, Attention), while "Propulsion(Attn)" applies it only to the Attention layer. Propulsion(All)^3 refers to three Propulsion mechanisms in each layer.

4.2 Language Model Performance

Datasets : For the GLUE Benchmark, we evaluate our *Propulsion* method on CoLA, SST-2, MRPC, STS-B, QQP, MNLI, QNLI, and RTE tasks of the GLUE Benchmarks(Wang et al., 2018). We also use SQuAD v1.1 (Rajpurkar et al., 2016) and SQuAD v2.0 (Rajpurkar et al., 2018) datasets to measure performance on question-answering tasks, and we use the XSum (Narayan et al., 2018) and CNN/DailyMail (Hermann et al., 2015) datasets to measure text summarization performance.

Model Selection & Hyperparameter : For the GLUE benchmark, the models we select for fine-tuning are RoBERTa-base (RoB_B) with 125M parameters and RoBERTa-large (RoB_L) with 355M parameters from Liu et al. (2019). We set the *Propulsion* degree to 15 as discussed in Section 2 for SST-2, QQP, RTE, and STS-B; 55 for QNLI and MRPC; and 20 for the other GLUE datasets.

For the SQuAD v1.1 and SQuAD v2.0 datasets, we employ DeBERTaV3-base (He et al., 2020). For both SQuAD v1.1 and SQuAD v2.0, we set the *Propulsion* degree to 35.

For the XSum and CNN/DailyMail datasets, we chose the BART-large model (Lewis et al., 2019) with 406M parameters. For XSum and CNN/DailyMail, we set the Propulsion degrees to 35 and 25.

Results : Table 1 shows the GLUE task validation results of *Propulsion*, in comparison with baselines, we can see that *Propulsion* can achieve better or on-par performance compared with existing PEFT approaches on the GLUE dataset but with much less trainable parameters. Overall, *Propulsion* exhibits enhancements of 2.48%, 3.15%, and 3.17% in accuracy over AdaLoRA, PROPETL Prefix, and (IA)^3, respectively, and 1.94%, 1.87%, and 8.92% improvements in the F1 score.

Table 2 compares the validation performance of *Propulsion* and other PEFT methods on question-answering and text summarization tasks. For question answering tasks, *Propulsion* outperforms the other PEFT methods on both the SQuAD datasets. *Propulsion* beats AdaLoRA, the second highest performing PEFT method, by 0.66 in EM and 0.51 in F1 score while being 7.89 times smaller in parameter size. Comparing to LoKr, which has the least number of trainable parameters amongst the baseline PEFT methods, *Propulsion* outperforms

PEFT Method	#TPs	SQuADv1.1	SQuADv2.0	#TPs	XSum	CNN/DailyMail
FT	460M	82.83 / 88.14	82.92 / 83.75	460M	40.73 / 16.19 / 30.13	39.16 / 18.92 / 37.04
Prompt tuning	0.155M	74.52 / 78.42	73.59 / 76.72	0.524M	38.24 / 14.46 / 27.89	37.42 / 17.43 / 34.92
Prefix-tuning	2.683M	78.38 / 82.94	74.94 / 79.04	4.482M	38.24 / 15.16 / 28.84	38.32 / 17.72 / 35.76
LoKr	0.089M	80.64 / 86.45	80.14 / 81.96	0.194M	39.03 / 16.14 / 30.42	39.12 / 17.98 / 37.75
Bitfit	0.161M	80.53 / 86.25	79.06 / 83.75	0.885M	39.10 / 16.87 / 30.43	39.93 / 18.12 / 38.85
LoHa	0.885M	81.43 / 88.02	81.67 / 85.01	1.769M	39.12 / 17.08 / 31.39	39.98 / 18.84 / 38.01
LoRA	0.442M	81.64 / 87.16	**82.76** / 85.75	1.763M	40.63 / **18.44** / **32.35**	**40.74** / <u>19.10</u> / <u>39.24</u>
AdaLoRA	0.663M	<u>81.16</u> / <u>87.75</u>	82.63 / **85.82**	2.655M	<u>40.95</u> / <u>18.28</u> / <u>31.84</u>	40.53 / 18.24 / **39.63**
Propulsion(All)	0.161M	**81.73** / **88.07**	<u>82.68</u>/ <u>85.81</u>	0.330M	**40.98** / 18.18 / 31.42	<u>40.56</u> / **19.28** / 38.76
Propulsion(Attn)	0.055M	80.95 / 87.20	81.02 / 85.50	0.110M	38.64 / 15.45 / 29.25	38.74 / 17.08 / 35.03

Table 2: Performance of DeBERTaV3-base and BART-large on SQuAD v1.1 and v2.0 benchmarks with EM/F1 and ROUGE scores (ROUGE-1/ROUGE-2/ROUGE-L). Here, the **bolded** values indicate the best performance, while the <u>underlined</u> values represent the second-best performance.

LoKr by 2.92 in EM and by 2.29 in F1-score while having fewer parameters.

For text summarization, *Propulsion* has the highest ROUGE-1 score among the baseline PEFT methods on both datasets. It also has the best ROUGE-2 score and the second-best ROUGE-L score on the CNN/DailyMail dataset. For XSum, LoRA and AdaLoRA have higher ROUGE-2 and ROUGE-L scores than *Propulsion*. This may be due to the limitations *Propulsion* may have by being constrained by the model's dimension, whereas LoRA and AdaLoRA have more flexibility with more parameters, which is evident by higher ROUGE-2/L scores. Despite this, *Propulsion's* performance on CNN/DailyMail shows that it achieves on-par performance with methods like LoRA and AdaLoRA while having a significantly smaller parameter size. For both tables, we used the validation set to test the performance.

4.3 Large Language Models Performance

Datasets : We perform a thorough evaluation using thirteen benchmark datasets, covering common sense reasoning and mathematical reasoning tasks.

For common sense reasoning, we employ a diverse range of datasets, including BoolQ (Clark et al., 2019), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2021), ARC-easy and ARC-challenge (Clark et al., 2018), and OBQA (Mihaylov et al., 2018), to ensure a comprehensive assessment of our model's ability to handle various facts of common sense reasoning.

For arithmetic reasoning tasks, we also use several professional datasets, including MultiArith (Roy and Roth, 2016), GSM8K (Cobbe et al., 2021), AddSub (Hosseini et al., 2014), SingleEq (Koncel-Kedziorski et al., 2015), and SVAMP (Patel et al., 2021) to evaluate the performance of our model on solving various different arithmetic reasoning-related problems.

Model Selection & Hyperparameters: For the commonsense and mathematical reasoning tasks described, we select several LLMs to fine-tune using both standard baselines and our proposed *Propulsion* methods for comparison.

The LLMs chosen include BLOOMz (7B parameters) (Muennighoff et al., 2022), GPT-J (6B parameters), LLaMA (7B parameters, denoted as $\textbf{LLaMA}_{\textbf{7B}}$), and LLaMA (13B parameters, denoted as $\textbf{LLaMA}_{\textbf{13B}}$). For BLOOMz, $\textbf{LLaMA}_{\textbf{7B}}$, $\textbf{LLaMA}_{\textbf{13B}}$, and $\textbf{GPT-J}_{\textbf{6B}}$, we set the *Propulsion* degree to 15 for both reasoning tasks. Additionally, we apply a dropout rate of 0.1 for both hidden layers and attention mechanisms, along with L2 regularization. Each model layer is fine-tuned using 5 distinct *Propulsion* parameters to assess the effectiveness of our approach.

Results : Table 3 shows the accuracy results on all four LLMs across the thirteen benchmarks. Across the board, *Propulsion* outperforms state-of-the-art PEFT methods on both commonsense and mathematical reasoning tasks. On average across the four LLMs tested on these benchmarks, *Propulsion* shows competitive performance on all of the benchmarks while maintaining the highest accuracy on benchmarks like GSM8K.Notably, fine-tuning the BLOOMz and GPT-J models demonstrates competitive performance against the baseline methods. For datasets like SIQA, and HellaSwag, our method achieves 1.33%, 0.97% improvement than the state-of-the-art PEFT method on accuracy. And for LLaMA model fine-tuning ($\textbf{LLaMA}_{\textbf{7B}}$, $\textbf{LLaMA}_{\textbf{13B}}$), *Propulsion* also reach better performance than other baselines on most datasets, e.g. on the AddSub and SVQMP datasets,

LLM	Method	BoolQ	PIQA	SIQA	H.Swag	W.Grande	ARC-e	ARC-c	OBQA	MultiArith	GSM8K	AddSub	SingleEq	SVAMP
BLOOMz$_{7B}$	Prefix	58.53	62.24	65.41	48.32	66.63	68.13	49.32	63.51	78.41	66.45	67.52	66.94	49.10
	AdaLoRA	64.94	**74.68**	72.49	52.89	68.30	73.21	56.59	72.85	79.43	70.25	68.93	70.93	53.89
	Parallel	63.30	73.33	71.01	52.50	71.60	69.45	54.14	68.60	78.90	70.17	70.33	70.84	53.95
	LoRA	65.89	73.92	73.33	56.65	71.39	**73.46**	57.15	72.31	**79.50**	70.93	70.90	70.59	53.85
	Propulsion	**66.38**	74.63	**74.62**	**57.25**	**72.33**	73.09	**57.61**	**73.12**	79.36	**70.95**	**70.92**	**71.22**	**54.52**
GPT-J$_{6B}$	Prefix	62.28	65.04	67.72	44.15	63.71	63.59	46.47	58.31	83.12	67.44	75.25	78.46	49.12
	AdaLoRA	65.19	67.58	**71.22**	45.16	**66.03**	64.10	**47.75**	63.92	88.51	72.45	80.21	**82.03**	56.14
	Parallel	63.17	67.91	68.97	45.79	66.06	62.42	45.32	60.42	89.11	72.04	80.50	81.50	55.43
	LoRA	65.50	67.63	69.46	45.60	66.37	63.56	46.81	63.82	88.30	72.22	80.60	81.24	56.63
	Propulsion	**65.97**	**68.05**	69.96	**45.99**	66.18	**64.45**	46.95	**64.56**	**89.19**	**72.82**	**81.41**	81.42	**56.68**
LLaMA$_{7B}$	Prefix	67.33	79.46	75.80	70.04	72.11	71.67	57.33	69.98	84.18	68.47	81.04	80.00	52.17
	AdaLoRA	67.03	78.69	76.06	75.85	76.47	76.26	60.36	74.22	89.81	77.07	86.70	83.01	60.25
	Parallel	65.02	78.10	**77.52**	75.57	76.78	75.48	60.54	74.02	90.20	76.13	86.55	**83.70**	59.16
	LoRA	67.09	79.37	76.15	**76.86**	**77.54**	76.54	60.55	74.63	90.13	75.68	84.67	82.14	59.94
	Propulsion	**68.99**	**79.47**	77.02	76.73	77.06	**76.64**	**61.29**	**74.76**	**90.21**	**77.57**	**87.63**	82.60	**60.51**
LLaMA$_{13B}$	Prefix	68.38	80.99	77.80	75.00	76.35	77.62	61.32	72.94	87.22	71.09	84.09	81.28	58.25
	AdaLoRA	71.71	82.55	78.88	90.60	83.01	83.04	67.33	**81.76**	90.55	**80.19**	87.00	87.10	66.03
	Parallel	71.39	83.33	78.32	**91.40**	83.24	83.34	66.43	80.99	90.88	79.24	**88.16**	87.08	65.63
	LoRA	71.19	83.99	**79.15**	90.86	83.24	83.35	67.05	81.37	90.27	78.90	86.89	86.07	65.85
	Propulsion	**71.93**	**84.12**	79.01	90.73	**83.60**	**83.44**	67.64	81.38	**90.91**	78.71	87.64	**87.11**	**66.67**

Table 3: Accuracy comparison of Commonsense and Mathematical reasoning performance across different PEFTs with 3% performance reduction.

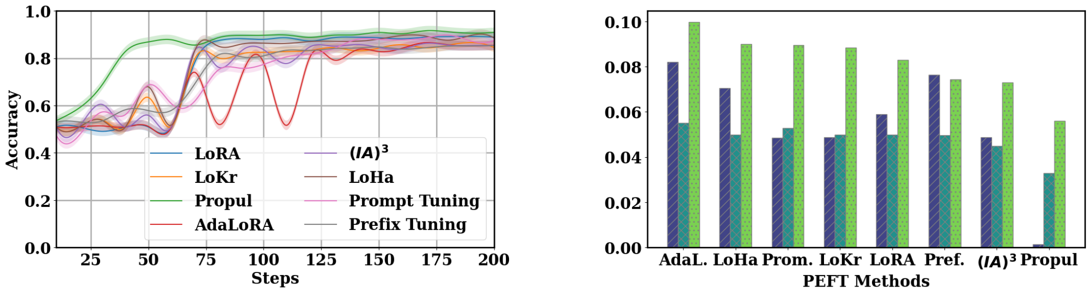


Figure 3: Comparative Analysis of PEFT Methods on the SST-2 Dataset. On the right-side graph, we shortened the following method names: AdaLoRA to AdaL., Prompt Tuning to Prom., Propulsion to Propul, and Prefix-Tuning to Pref. In this graph, purple represents the percentage of parameters after applying these methods, the cyan represents the total training time in hours, and the green represents the iteration time in seconds.

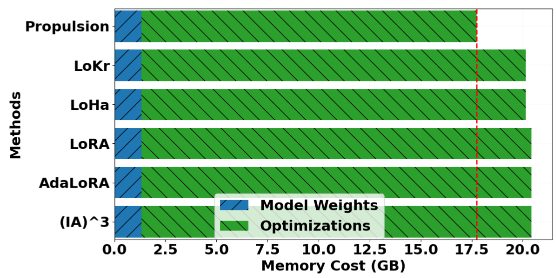


Figure 4: Memory Cost Comparison of PEFT Methods. The blue bars represent the memory cost of the original model weights, whereas the green bars represent the optimization memory cost for each of these methods.

Methods	Space	Time	#TPs
Propulsion	$O(d)$	$O(d)$	d
FT	$O(d \times d)$	$O(d \times d)$	d^2
$(IA)^3$	$O(d_k + d_v + d_{ff})$	$O(d_k + d_v + d_{ff})$	$3d$
Prompt	$O(d \times l_p)$	$O(d \times l_p)$	$l_p.d$
Prefix	$O(L \times d \times l_p)$	$O(L \times d \times l_p)$	$L.l_p.d$
LoRA	$O((d+d) \times r)$	$O((d+d) \times r)$	$2dr$
LoRA-FA	$O((d+d) \times r)$	$O((d+d) \times r)$	dr
AdaLoRA	$O((d+d+r) \times r)$	$O((d+d+r) \times r)$	$2dr + r^2$
LoHa	$O(2r \times (d+d))$	$O(2r \times (d+d))$	$4dr$

Table 4: Space/Time Complexity and Total Trainable Parameters (#TPs) for *Propulsion* method and baseline methods for single layer $W \in \mathbb{R}^{d \times d}$. Within this table, we define d_k, d_v, and d_{ff} as the dimensions of three learned vectors in $(IA)^3$; and l_p as the length of the prompt added to the input/layers in prompt tuning and prefix-tuning. For LoRA-type methods, we use r to represent the rank dimensions.

Propulsion shows enhancements of 0.97% and 0.66% in accuracy over the state-of-the-art PEFT method. While maintaining or improving accuracy, *Propulsion* also has a much smaller percentage of total parameters. Additional experiments on LLMs are described in Appendix K.

4.4 Efficiency Comparison

Our study evaluates diverse PEFT techniques on their performance, training efficiency, and memory usage. We conduct these experiments using the SST-2 dataset, divided into 64 batches. We train

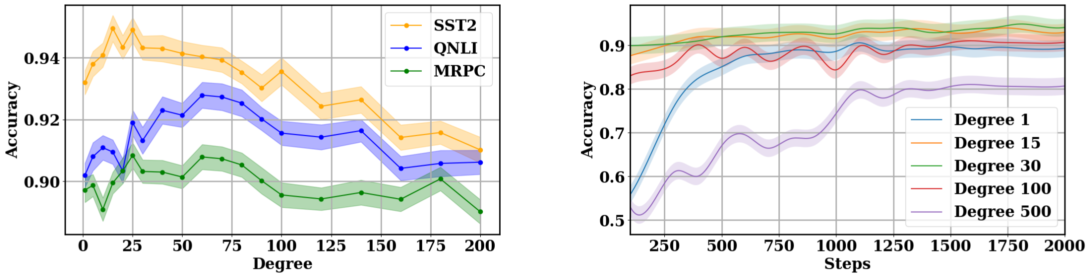


Figure 5: Left: performance vs. degree for SST-2, QNLI, and MRPC. Right: training steps vs. accuracy for SST-2.

on a H100 with 80 GB of memory. The default parameters include the learning rate of 1×10^{-4}, the weight decay of 0.02, dropout 0.1.

Training efficiency: Figure 3 illustrates the training convergence of our models and baselines. On the left side of the figure, it shows that our *Propulsion* model exhibits a fast convergence and achieves a higher accuracy of 0.9 in just 50 iterations, whereas the baseline AdaLoRA method requires approximately 200 iterations to attain an accuracy of 0.87, and the LoRA method requires almost 75 iterations to reach an accuracy comparable to that of *Propulsion*. Furthermore, the other methods, including LoKr, $(IA)^3$, and LoHa, require more than 150 iterations to achieve an accuracy of 0.8.

Parameter Efficiency : In terms of parameters, we present the efficiency of each method in Tables 1 and 2, as well as a graphical representation in Figure 3 (Right). It is clear that *Propulsion* demonstrates superior efficiency in terms of faster training time, and reduced memory usage because of its parameter reduction. Table 4 compares the space/time complexities and total trainable parameters of our *Propulsion* method to other baseline PEFT methods.

Memory Efficiency: In terms of memory efficiency of the GPU, as illustrated in Figure 4, *Propulsion* consumes only approximately 17.2 GB of GPU memory for training, including model weights and optimization. In comparison, other baseline methods consume more than 20.0 GB of GPU memory, making *Propulsion* approximately 1.5 times more memory-efficient than other PEFT methods. Additionally, in Appendix D (Table 6), we present a comparison of delta weight reparameterization methods for the backward pass during optimization

Layer	SST-2	MRPC	QQP	QNLI	RTE	Params
Embedding	73.45	70.32	75.28	79.38	66.3	0.0115M
MLP	92.42	86.42	82.38	89.35	72.43	0.0115M
Key	92.52	86.84	83.95	88.14	72.19	0.0115M
Value	92.68	86.59	83.05	88.76	73.93	0.0115M
Query	91.53	86.99	83.84	89.28	73.68	0.0115M
K+Q+V	92.72	89.01	85.82	89.89	75.02	0.0283M
All	93.18	89.34	89.11	92.79	75.66	0.0861M

Table 5: Accuracy [%] and the parameter size for different layer configurations with *Propulsion* across datasets.

4.5 Ablation Study

Propulsion Degree Initialization: In this section, we explore the impact of the *Propulsion* degree as a hyperparameter on model performance across different datasets. Figure 5 (left) shows the accuracy on SST-2, QNLI, and MRPC for degrees ranging from 0 to 200. SST-2 achieves its highest accuracy of 95% at a degree of 25, while QNLI peaks at 94% between 50 and 75 degrees, and MRPC at 92% around 25 degrees. After reaching peak accuracy, both QNLI and MRPC show a decline, indicating overfitting as the *Propulsion* degree increases.

Figure 5 (right) shows the training dynamics on SST-2. Lower degrees (1 and 15) converge faster, achieving high accuracy early, while higher degrees (100 and 500) take longer. By 2000 steps, all degrees converge, but lower degrees stabilize faster, suggesting they are more effective for rapid learning, with higher degrees needing more steps for similar performance.

Positional Impact of Propulsion: Table 5 shows an ablation analysis of *Propulsion* configured across various layers, including embedding, MLP, Key (K), Query(Q), Value (V), and different combinations of layers. Adding *Propulsion* to the attention mechanism (K + V + Q) achieved an accuracy of 93.72% on the SST-2 dataset. When examined individually, we obtained accuracies of 91.52%, 92.52%, and 92.68% in the Query, and Value, respectively. However, *Propulsion* in the embedding layer does not yield performance comparable to that of the other layers. Nonetheless, *Propulsion* in all layers leads to substantial accuracy improvements of 94.89%, 90.52%, 90.86%, 92.79%, and 77.60% for the SST-2, MRPC, QQP, QNLI, and RTE datasets, respectively.

Additional ablation studies are described in Appendix F.

5 Related Work

The development of parameter-efficient fine-tuning (PEFT) techniques is essential in NLP due to the increasing complexity of LLMs. These techniques enhance performance while reducing computational and memory requirements, as demonstrated by recent studies (Liu et al., 2022a; Nguyen et al., 2023; Chow et al., 2024). PEFT techniques have been proven effective across a wide range of NLP tasks, including (Fu et al., 2023; He et al., 2021). Previous research (Liu et al., 2021b, 2023; Zhang et al., 2023; Hu et al., 2021; Li and Liang, 2021; Zaken et al., 2021) has shown that PEFT techniques can significantly improve the performance of LLMs while utilizing low resources.

Prompt Tuning entails adding learnable parameters as virtual tokens at the model's input (Lester et al., 2021) or within each layer (Li and Liang, 2021). Recent advancements have refined these methods for NLU (Liu et al., 2021b) and NLG (An et al., 2022), including adding residual connections for stability (Razdaibiedina et al., 2023b) and adapting to continual learning (Razdaibiedina et al., 2023a). Innovative techniques like MixPAVE (Yang et al., 2023a) and E2VPT (Han et al., 2023) integrate input and value prompts to boost performance. These methods have significantly enhanced specific NLP tasks such as text classification, machine translation, and dialogue generation.

Low-Rank Adaptation (LoRA), introduced by Hu et al. (2021), is a memory-efficient fine-tuning technique extensively studied. Renduchintala et al. (2023), Sheng et al. (2023), and Xia et al. (2024) explored its multitask learning potential. Wang et al. (2023) showed practical applications, while Dettmers et al. (2024) optimized memory usage. Lialin et al. (2023) proposed ReLoRA, requiring a full-rank warm-up. Adaptive methods by Zhang et al. (2023) dynamically adjust low-rank parameters. Edalati et al. (2022) introduced the Low-Rank Kronecker Product (LoKr), and Shi et al. (2024) developed ResLoRA with residual paths. Hyeon-Woo et al. (2021) presented the Low-Rank Hadamard Product (LoHa), while Qiu et al. (2024) and Liu et al. (2024) introduced Orthogonal Fine-tuning (OFT) and OFT with butterfly factorization (BOFT), using orthogonal matrices to modify pre-trained weights, enhancing fine-tuning efficiency and performance.

Unlike previous PEFT approaches, we propose a new concept of adaptive *Propulsion* that changes the output direction of the model by Propulsion a force to achieve task-specific goals. We adjust the *Propulsion* parameter during the training process, which decides how much push needs to change the direction. (More details related work in Appendix J)

6 Conclusion

Fine-tuning extensive language models can be costly in terms of hardware and storage switching expenses, and the financial investment required to host separate instances of diverse tasks is often substantial. We propose *Propulsion*, a parameter-efficient fine-tuning method that adds trainable *Propulsion* parameters to each layer while keeping the original parameters frozen. The goal of *Propulsion* is to achieve task-specific objectives without modifying the original parameters of the LLMs. Our experiments on natural language processing, question answering, text summarization, common sense reasoning, and mathematical reasoning show that *Propulsion* outperforms existing methods in terms of accuracy, efficiency, faster convergence, reduced training time, and lower memory usage. Our results demonstrate that *Propulsion* outperforms current PEFT techniques while requiring fewer trainable parameters. For example, *Propulsion* uses 37 times fewer parameters than AdaLoRA and achieves 4.05% higher accuracy.

7 Limitations

The *Propulsion* method has a few limitations. First, it offers limited control over the model compared to other methods such as LoRA, which allows adjustments through changes in rank. In *Propulsion*, the ability to steer a model is constrained by the number of dimensions in each layer. Essentially, we can only adjust the *Propulsion* parameters equal to the number of dimensions of a layer, which restricts the extent to which we can tweak the model's behavior. Additionally, since each parameter in the *Propulsion* method works independently without influencing others, it may be harder to make coordinated changes across the model. Moreover, the success of *Propulsion* depends on the quality of the pre-trained language model.

References

Zahra Rahimi Afzal, Tara Esmaeilbeig, Mojtaba Soltanalian, and Mesrob I Ohannessian. Can the spectrum of the neural tangent kernel anticipate fine-tuning performance? In *Adaptive Foundation Models: Evolving AI for Personalized and Efficient Learning*.

Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malartic, et al. 2023. The falcon series of open language models. *arXiv preprint arXiv:2311.16867*.

Shengnan An, Yifei Li, Zeqi Lin, Qian Liu, Bei Chen, Qiang Fu, Weizhu Chen, Nanning Zheng, and Jian-Guang Lou. 2022. Input-tuning: Adapting unfamiliar inputs to frozen pretrained models. *arXiv preprint arXiv:2203.03131*.

Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. 2021. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*, pages 610–623.

Jonathan Bisk et al. 2020. Piqa: Reasoning about physical commonsense in natural language. *Proceedings of XYZ Journal*, 5(2):123–134.

Vitalii Budashko. 2020. Thrusters physical model formalization with regard to situational and identification factors of motion modes. In *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)*, pages 1–6. IEEE.

Arslan Chaudhry, Naeemullah Khan, Puneet Dokania, and Philip Torr. 2020. Continual learning in low-rank orthogonal subspaces. *Advances in Neural Information Processing Systems*, 33:9900–9911.

Han Chen, Garvesh Raskutti, and Ming Yuan. 2019. Non-convex projected gradient descent for generalized low-rank tensor regression. *Journal of Machine Learning Research*, 20(5):1–37.

Lei Chen, Houwei Chou, and Xiaodan Zhu. 2022a. Developing prefix-tuning models for hierarchical text classification. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 390–397.

Yaofo Chen, Yong Guo, Daihai Liao, Fanbing Lv, Hengjie Song, and Mingkui Tan. 2022b. Automatic subspace evoking for efficient neural architecture search. *arXiv preprint arXiv:2210.17180*.

Yudong Chen and Martin J Wainwright. 2015. Fast low-rank estimation by projected gradient descent: General statistical and algorithmic guarantees. *arXiv preprint arXiv:1509.03025*.

Kingsum Chow, Yu Tang, Zhiheng Lyu, Anil Rajput, and Khun Ban. 2024. Performance optimization in the llm world 2024. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering*, pages 156–157.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems, 2021. *URL https://arxiv.org/abs/2110.14168*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ali Edalati, Marzieh Tahaei, Ivan Kobyzev, Vahid Partovi Nia, James J Clark, and Mehdi Rezagholizadeh. 2022. Krona: Parameter efficient tuning with kronecker adapter. *arXiv preprint arXiv:2212.10650*.

Zihao Fu, Haoran Yang, Anthony Man-Cho So, Wai Lam, Lidong Bing, and Nigel Collier. 2023. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Robert M. Gray and David L. Neuhoff. 1998. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383.

Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. 2018. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*.

Cheng Han, Qifan Wang, Yiming Cui, Zhiwen Cao, Wenguan Wang, Siyuan Qi, and Dongfang Liu. 2023. Eˆ 2vpt: An effective and efficient approach for visual prompt tuning. *arXiv preprint arXiv:2307.13770*.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. *arXiv preprint arXiv:2110.04366*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. *Advances in neural information processing systems*, 28.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Nam Hyeon-Woo, Moon Ye-Bin, and Tae-Hyun Oh. 2021. Fedpara: Low-rank hadamard product for communication-efficient federated learning. *arXiv preprint arXiv:2108.06098*.

Arthur Jacot, Franck Gabriel, and Clément Hongler. 2018. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31.

Uijeong Jang, Jason D Lee, and Ernest K Ryu. 2024. Lora training in the ntk regime has no spurious local minima. *arXiv preprint arXiv:2402.11867*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Jeonghoon Kim, Jung Hyun Lee, Sungdong Kim, Joonsuk Park, Kang Min Yoo, Se Jung Kwon, and Dongsoo Lee. 2024. Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. *Advances in Neural Information Processing Systems*, 36.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

Md Kowsher, Tara Esmaeilbeig, Chun-Nam Yu, Mojtaba Soltanalian, and Niloofar Yousefi. 2024. Rocoft: Efficient finetuning of large language models with row-column updates. *arXiv preprint arXiv:2410.10075*.

Md Kowsher, Md Shohanur Islam Sobuj, Asif Mahmud, Nusrat Jahan Prottasha, and Prakash Bhat. 2023. L-tuning: Synchronized label tuning for prompt and prefix in llms. *arXiv preprint arXiv:2402.01643*.

Brett W Larsen, Stanislav Fort, Nic Becker, and Surya Ganguli. 2021. How many degrees of freedom do we need to train deep networks: a loss landscape perspective. *arXiv preprint arXiv:2107.05802*.

Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2023. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.

Yoonho Lee and Seungjin Choi. 2018. Gradient-based meta-learning with learned layerwise metric and subspace. In *International Conference on Machine Learning*, pages 2927–2936. PMLR.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Bingrui Li, Jianfei Chen, and Jun Zhu. 2024. Memory efficient optimizers with 4-bit states. *Advances in Neural Information Processing Systems*, 36.

Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. 2023. Relora: High-rank training through low-rank updates. In *Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ NeurIPS 2023)*.

Zhaojiang Lin, Andrea Madotto, and Pascale Fung. 2020. Exploring versatile generative language model via parameter-efficient transfer learning. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 441–459, Online. Association for Computational Linguistics.

Haokun Liu, Derek Tam, Mohammed Muqeeth, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin A Raffel. 2022a. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. *Advances in Neural Information Processing Systems*, 35:1950–1965.

Peiyu Liu, Ze-Feng Gao, Wayne Xin Zhao, Zhi-Yuan Xie, Zhong-Yi Lu, and Ji-Rong Wen. 2021a. Enabling lightweight fine-tuning for pre-trained language model compression based on matrix product operators. *arXiv preprint arXiv:2106.02205*.

Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, Yandong Wen, Michael J. Black, Adrian Weller, and Bernhard Schölkopf. 2024. Parameter-efficient orthogonal finetuning via butterfly factorization. *Preprint*, arXiv:2311.06243.

Xiao Liu, Heyan Huang, Ge Shi, and Bo Wang. 2022b. Dynamic prefix-tuning for generative template-based event extraction. *arXiv preprint arXiv:2205.06166*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2021b. P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks. *arXiv preprint arXiv:2110.07602*.

Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang, and Jie Tang. 2023. Gpt understands, too. *AI Open*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. `https://github.com/huggingface/peft`.

Ben Mann, N Ryder, M Subbiah, J Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, A Askell, S Agarwal, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.

Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le Scao, M Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, et al. 2022. Crosslingual generalization through multitask finetuning. *arXiv preprint arXiv:2211.01786*.

Shashi Narayan, Shay B Cohen, and Mirella Lapata. 2018. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. *arXiv preprint arXiv:1808.08745*.

Minh Nguyen, KC Kishan, Toan Nguyen, Ankit Chadha, and Thuy Vu. 2023. Efficient fine-tuning large language models for knowledge-aware response planning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 593–611. Springer.

Elvis Nunez, Maxwell Horton, Anish Prabhu, Anurag Ranjan, Ali Farhadi, and Mohammad Rastegari. 2023. Lcs: Learning compressible subspaces for efficient, adaptive, real-time network compression at inference time. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3818–3827.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Zeju Qiu, Weiyang Liu, Haiwen Feng, Yuxuan Xue, Yao Feng, Zhen Liu, Dan Zhang, Adrian Weller, and Bernhard Schölkopf. 2024. Controlling text-to-image diffusion by orthogonal finetuning. *Preprint*, arXiv:2306.07280.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Mercy Ranjit, Gopinath Ganapathy, Shaury Srivastav, Tanuja Ganu, and Srujana Oruganti. 2024. Rad-phi2: Instruction tuning phi-2 for radiology. *arXiv preprint arXiv:2403.09725*.

Anastasia Razdaibiedina, Yuning Mao, Rui Hou, Madian Khabsa, Mike Lewis, and Amjad Almahairi. 2023a. Progressive prompts: Continual learning for language models. *arXiv preprint arXiv:2301.12314*.

Anastasia Razdaibiedina, Yuning Mao, Rui Hou, Madian Khabsa, Mike Lewis, Jimmy Ba, and Amjad Almahairi. 2023b. Residual prompt tuning: Improving prompt tuning with residual reparameterization. *arXiv preprint arXiv:2305.03937*.

Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. 2023. Tied-lora: Enhacing parameter efficiency of lora with weight tying. *arXiv preprint arXiv:2311.09578*.

Matthias C Rillig, Marlene Ågerstrand, Mohan Bi, Kenneth A Gould, and Uli Sauerland. 2023. Risks and benefits of large language models for the environment. *Environmental Science & Technology*, 57(9):3464–3466.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning*, pages 4596–4604. PMLR.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, et al. 2023. S-lora: Serving thousands of concurrent lora adapters. *arXiv preprint arXiv:2311.03285*.

Shuhua Shi, Shaohan Huang, Minghui Song, Zhoujun Li, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, and Qi Zhang. 2024. Reslora: Identity residual mapping in low-rank adaption. *arXiv preprint arXiv:2402.18039*.

Akiyoshi Tomihari and Issei Sato. 2024. Understanding linear probing then fine-tuning language models from ntk perspective. *arXiv preprint arXiv:2405.16747*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Peter J Turchi. 1998. *Propulsion techniques: action and reaction*. AIAA.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Yiming Wang, Yu Lin, Xiaodong Zeng, and Guannan Zhang. 2023. Multilora: Democratizing lora for better multi-task learning. *arXiv preprint arXiv:2311.11501*.

Zihan Wang, Peiyi Wang, Tianyu Liu, Binghuai Lin, Yunbo Cao, Zhifang Sui, and Houfeng Wang. 2022. Hpt: Hierarchy-aware prompt tuning for hierarchical text classification. *arXiv preprint arXiv:2204.13413*.

Tongtong Wu, Linhao Luo, Yuan-Fang Li, Shirui Pan, Thuy-Trang Vu, and Gholamreza Haffari. 2024. Continual learning for large language models: A survey. *arXiv preprint arXiv:2402.01364*.

Wenhan Xia, Chengwei Qin, and Elad Hazan. 2024. Chain of lora: Efficient fine-tuning of language models via residual learning. *arXiv preprint arXiv:2401.04151*.

Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. 2023. Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment. *arXiv preprint arXiv:2312.12148*.

Li Yang, Qifan Wang, Jingang Wang, Xiaojun Quan, Fuli Feng, Yu Chen, Madian Khabsa, Sinong Wang, Zenglin Xu, and Dongfang Liu. 2023a. Mixpave: Mix-prompt tuning for few-shot product attribute value extraction. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 9978–9991.

Xianjun Yang, Wei Cheng, Xujiang Zhao, Wenchao Yu, Linda Petzold, and Haifeng Chen. 2023b. Dynamic prompting: A unified framework for prompt tuning. *arXiv preprint arXiv:2303.02909*.

Elad Ben Zaken, Shauli Ravfogel, and Yoav Goldberg. 2021. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. *arXiv preprint arXiv:2106.10199*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Guangtao Zeng, Peiyuan Zhang, and Wei Lu. 2023. One network, many masks: Towards more parameter-efficient transfer learning. *arXiv preprint arXiv:2305.17682*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2022. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*.

Teng Zhang and Xing Fan. 2024. Projected gradient descent algorithm for low-rank matrix estimation. *arXiv preprint arXiv:2403.02704*.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. Galore: Memory-efficient llm training by gradient low-rank projection. *arXiv preprint arXiv:2403.03507*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

A Training Dynamics of Propulsion Explained by NTK

Theorem 1 Let $\phi_P(\mathbf{x};\boldsymbol{\theta}_t)$ be the output of the Propulsion model at time step t, where the base matrix $\boldsymbol{\theta}_0$ is pre-trained and fixed, and the diagonal matrix $\mathbf{Z}_t$ is updated during training. Let $\phi_F(\mathbf{x};\boldsymbol{\theta}_t)$ be the output of the fully fine-tuned model at time step t.

Under the NTK regime, where the width d of the network is sufficiently large, the NTK for Propulsion fine-tuning approximates the NTK for full fine-tuning with high probability. Formally, for inputs $\mathbf{x},\mathbf{x}' \in \mathbb{R}^d$, the NTK for Propulsion satisfies:

$$\mathbf{K}^F\left(\mathbf{x},\mathbf{x}'\right) \approx \mathbf{K}^P\left(\boldsymbol{\theta}_0\mathbf{x}_i,\boldsymbol{\theta}_0\mathbf{x}_j\right)$$

Furthermore, the error between the NTK for Propulsion and the NTK for full fine-tuning can be bounded using the Johnson-Lindenstrauss Lemma. Specifically, for any $\varepsilon > 0$ and constant c, with high probability:

$$\Pr\left[\left|(\boldsymbol{\theta}_0\mathbf{x}_i)^\top(\boldsymbol{\theta}_0\mathbf{x}_j) - \mathbf{x_i}^\top\mathbf{x_j}\right|\right] \geq 1-4\exp\left(-\frac{(\varepsilon^2-\varepsilon^3)d}{4}\right)$$

To establish the proof of the theorem, we first introduce the definitions of the NTK Kernel and the Kernel Behavior specific to the Propulsion method.

Definition-1 (NTK Kernel): Let $\mathbf{K}(\mathbf{x},\mathbf{x}')$ represent the Neural Tangent Kernel (NTK) of a model. The kernel is defined as the inner product of the gradients of the model outputs with respect to the parameters $\boldsymbol{\theta}$. Formally, for inputs $\mathbf{x},\mathbf{x}' \in \mathbb{R}^d$, the kernel is given by:

$$\mathbf{K}(\mathbf{x},\mathbf{x}') = \nabla_{\boldsymbol{\theta}}\phi_P(\mathbf{x};\boldsymbol{\theta})^\top \nabla_{\boldsymbol{\theta}}\phi_P(\mathbf{x}';\boldsymbol{\theta}),$$

where $\nabla_{\boldsymbol{\theta}}\phi_P(\mathbf{x};\boldsymbol{\theta})$ represents the gradient of the model output $\phi_P(\mathbf{x};\boldsymbol{\theta})$ with respect to the parameters $\boldsymbol{\theta}$.

Definition-2 (Kernel Behavior): Let $\boldsymbol{\theta}_t$ represent the parameters of a model at time step t, and let $\mathbf{x}$ be an arbitrary fixed input. The Propulsion model exhibits *kernel behavior* if the following properties are satisfied:

1. **Linearization**: The change in the model's output can be well-approximated by the first-order Taylor expansion. Specifically:

$$\phi_P(\mathbf{x};\boldsymbol{\theta}_t) - \phi_P(\mathbf{x};\boldsymbol{\theta}_{t-1}) \approx \langle \nabla\phi_P(\mathbf{x};\boldsymbol{\theta}_{t-1}), \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}\rangle,$$

   where $\nabla\phi_P(\mathbf{x};\boldsymbol{\theta})$ is the gradient of the model's output with respect to the parameters $\boldsymbol{\theta}$.

2. **Fixed Features**: The gradient of the model at time step t is approximately the same as the gradient at initialization, i.e.,

$$\nabla\phi_P(\mathbf{x};\boldsymbol{\theta}_t) \approx \nabla\phi_P(\mathbf{x};\boldsymbol{\theta}_0),$$

   where $\boldsymbol{\theta}_0$ refers to the parameters at initialization.

Proof: Let $\boldsymbol{\theta}_t$ represent the parameters of the network at time step t, and $\phi_{\boldsymbol{\theta}}$ denote the output of the pre-trained network. Under the NTK approximation, the change in the network's output can be expressed as a first-order Taylor expansion:

$$\phi_{\boldsymbol{\theta}_{t+1}}(\mathbf{x}) \approx \phi_{\boldsymbol{\theta}_t}(\mathbf{x}) + \langle \nabla_{\boldsymbol{\theta}_t}\phi_{\boldsymbol{\theta}_t}(\mathbf{x}), \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\rangle.$$

In this work, we aim to analyze the Propulsion fine-tuning method in the context of NTK, and show that the NTK of Propulsion closely approximates the NTK of full fine-tuning.

Kernel Behavior: In stochastic gradient descent (SGD), the update to the parameters at step t is given by:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t &= -\eta\mathbb{E}_{\mathbf{x}\sim\mathscr{D}}[\nabla_{\boldsymbol{\theta}_t}\mathscr{L}(\phi_{\boldsymbol{\theta}_t}(\mathbf{x}))] \quad (4)\\ &= -\eta\mathbb{E}_{\mathbf{x}\sim\mathscr{D}}\left[\nabla_{\boldsymbol{\theta}_t}\phi_{\boldsymbol{\theta}_t}(\mathbf{x})\mathscr{L}'(\phi_{\boldsymbol{\theta}_t}(\mathbf{x}))\right] \quad (5)\end{aligned}$$

where $\mathscr{L}(\phi_{\boldsymbol{\theta}_t}(\mathbf{x}))$ represents the loss function and η is the learning rate.

The change in the output of the network at step t can be expressed as:

$$\begin{aligned}\nabla\boldsymbol{\theta}(\mathbf{x}') &= \phi_{\boldsymbol{\theta}_{t+1}}(\mathbf{x}') - \phi_{\boldsymbol{\theta}_t}(\mathbf{x}') \quad (6)\\ &= \langle \nabla_{\boldsymbol{\theta}_t}\phi_{\boldsymbol{\theta}_t}(\mathbf{x}'), \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\rangle \quad (7)\\ &= -\eta\nabla_{\boldsymbol{\theta}_t}\phi_{\boldsymbol{\theta}_t}(\mathbf{x}')^\top\mathbb{E}_{\mathbf{x}}\left[\nabla_{\boldsymbol{\theta}_t}\phi_{\boldsymbol{\theta}_t}(\mathbf{x})\mathscr{L}'(\phi_{\boldsymbol{\theta}_t}(\mathbf{x}))\right] \quad (8)\\ &= -\eta\mathbb{E}_{\mathbf{x}}\left[\nabla_{\boldsymbol{\theta}_t}\phi_{\boldsymbol{\theta}_t}(\mathbf{x}')^\top\nabla_{\boldsymbol{\theta}_t}\phi_{\boldsymbol{\theta}_t}(\mathbf{x})\mathscr{L}'(\phi_{\boldsymbol{\theta}_t}(\mathbf{x}))\right] \quad (9)\\ &= -\eta\mathbb{E}_{\mathbf{x}}\left[\mathbf{K}(\mathbf{x},\mathbf{x}')\mathscr{L}'(\phi_{\boldsymbol{\theta}_t}(\mathbf{x}))\right] \quad (10)\end{aligned}$$

where $\mathbf{K}(\mathbf{x},\mathbf{x}') = \nabla_{\boldsymbol{\theta}_t}\phi_{\boldsymbol{\theta}_t}(\mathbf{x})^\top\nabla_{\boldsymbol{\theta}_t}\phi_{\boldsymbol{\theta}_t}(\mathbf{x}')$ is the NTK matrix at time t.

We now proceed to prove by induction that the NTK of the Propulsion method closely approximates the NTK of full fine-tuning. In theory, we introduce a diagonal matrix $\mathbf{Z}$, and we can write the Propulsion model as:

$$\phi_P(\mathbf{x};\boldsymbol{\theta}) = \boldsymbol{\theta}_0\mathbf{x}\odot\mathbf{z} = \boldsymbol{\theta}_0\mathbf{x}\mathbf{Z},$$

where $\mathbf{Z}$ is a diagonal matrix, and the diagonal elements of $\mathbf{Z}$ correspond to the Propulsion parameters $\mathbf{z}$.

Base Case: Consider the model before training at $t = t_0$. The output of the Propulsion model can be written as:

$$\phi_P(\mathbf{x};\boldsymbol{\theta}_{t_0}) = \boldsymbol{\theta}_0\mathbf{x}\mathbf{Z}_0,$$

where $\boldsymbol{\theta}_0$ is the pre-trained weight matrix and $\mathbf{Z}_0 = I_n$ is the identity matrix (i.e., initially, the diagonal matrix $\mathbf{Z}$ is an identity matrix). In this case, the gradient with respect to the parameters is:

$$\nabla\phi_P(\mathbf{x};\boldsymbol{\theta}_{t_0}) = \boldsymbol{\theta}_0\mathbf{x}.$$

Since $\mathbf{Z}_0 = I_n$, the gradient is identical to the gradient of the fully fine-tuned model:

$$\nabla\phi_P(\mathbf{x};\boldsymbol{\theta}_{t_0}) = \nabla\phi_F(\mathbf{x};\boldsymbol{\theta}_{t_0}).$$

Thus, the NTK for Propulsion at initialization is identical to the NTK for full fine-tuning:

$$\mathbf{K}_P(\mathbf{x},\mathbf{x}') = \mathbf{K}_F(\mathbf{x},\mathbf{x}').$$

Inductive Hypothesis: Assume that at step t, the Propulsion model is of the form:

$$\phi_P(\mathbf{x};\boldsymbol{\theta}_t) = \boldsymbol{\theta}_0\mathbf{x}\mathbf{Z}_t,$$

where $\mathbf{Z}_t$ is the updated diagonal matrix at time t. The gradient with respect to the diagonal parameters is:

$$\nabla\phi_P(\mathbf{x};\boldsymbol{\theta}_t) = \nabla\phi_P(\boldsymbol{\theta}_0\mathbf{x};\mathbf{Z}_t).$$

We now compute the NTK for Propulsion at step t:

$$\nabla\phi_P(\mathbf{x}_i;\boldsymbol{\theta}_t)\cdot\nabla\phi_P(\mathbf{x}_j;\boldsymbol{\theta}_t)^\top = \nabla\phi_P(\boldsymbol{\theta}_0\mathbf{x}_i;\boldsymbol{\theta}_{z_t})\cdot\nabla\phi_P(\boldsymbol{\theta}_0\mathbf{x}_j;\boldsymbol{\theta}_{z_t})^\top \tag{11}$$

Now from the definition of NTK, we can write:

$$\nabla\phi_P(\mathbf{x}_i;\boldsymbol{\theta}_t)\cdot\nabla\phi_P(\mathbf{x}_j;\boldsymbol{\theta}_t)^\top = \mathbf{K}^P(\boldsymbol{\theta}_0\mathbf{x}_i,\boldsymbol{\theta}_0\mathbf{x}_j) \tag{12}$$

Inductive Step: We now show that the NTK for Propulsion converges to the NTK for full fine-tuning.

From the definition of kernel behavior in the NTK regime, we know that for large d, the width of the network, the change in the NTK over time is small. Specifically, for large d, we have:

$$\nabla\phi_P(\mathbf{x};\boldsymbol{\theta}_t) - \nabla\phi_P(\mathbf{x};\boldsymbol{\theta}_{t_0}) \approx \nabla\phi_F(\mathbf{x};\boldsymbol{\theta}_t) - \nabla\phi_F(\mathbf{x};\boldsymbol{\theta}_{t_0}).$$

Since at t_0, we have $\nabla\phi_P(\mathbf{x};\boldsymbol{\theta}_{t_0}) = \nabla\phi_F(\mathbf{x};\boldsymbol{\theta}_{t_0})$, it follows that:

$$\nabla\phi_P(\mathbf{x};\boldsymbol{\theta}_t) \approx \nabla\phi_F(\mathbf{x};\boldsymbol{\theta}_t). \tag{13}$$

Thus we can write

$$\nabla\phi_F(\mathbf{x}_i;\boldsymbol{\theta}_t)\cdot\nabla\phi_F(\mathbf{x}_j;\boldsymbol{\theta}_t)^\top \approx \mathbf{K}^P(\boldsymbol{\theta}_0\mathbf{x}_i,\boldsymbol{\theta}_0\mathbf{x}_j) \tag{14}$$

Which is simply implies

$$\mathbf{K}^F(\mathbf{x},\mathbf{x}') \approx \mathbf{K}^P(\boldsymbol{\theta}_0\mathbf{x}_i,\boldsymbol{\theta}_0\mathbf{x}_j) \tag{15}$$

Thus, the NTK for Propulsion approximates the NTK for full fine-tuning:

$$\mathbf{K}^F(\mathbf{x},\mathbf{x}') \approx \mathbf{K}^P(\boldsymbol{\theta}_0\mathbf{x}_i,\boldsymbol{\theta}_0\mathbf{x}_j)$$

Error Bound: To formalize the error between the NTK for Propulsion and full fine-tuning, we apply the Johnson-Lindenstrauss Lemma.

Given vectors $\mathbf{u},\mathbf{v}\in\mathbb{R}^d$ with $\|\mathbf{u}\|,\|\mathbf{v}\|\le c$, and a random matrix $A\in\mathbb{R}^{d\times k}$ with i.i.d. entries, the lemma states:

$$\Pr\left[\left|(A\mathbf{u})^\top(A\mathbf{v}) - \mathbf{u}^\top\mathbf{v}\right| \ge c\varepsilon\right] \le 4\exp\left(-\frac{(\varepsilon^2-\varepsilon^3)d}{4}\right).$$

Using the Johnson-Lindenstrauss lemma, with a probability of at least $1-4\exp\left(-\frac{(\varepsilon^2-\varepsilon^3)d}{4}\right)$ Applying this to our NTK matrices, we get:

$$\Pr\left[\left|(\boldsymbol{\theta}_0\mathbf{x}_i)^\top(\boldsymbol{\theta}_0\mathbf{x}_j) - \mathbf{x_i}^\top\mathbf{x_j}\right|\right] \ge 1-4\exp\left(-\frac{(\varepsilon^2-\varepsilon^3)d}{4}\right)$$

B Empirical Validation of NTK Approximation

In this section, we present empirical evidence to support the theoretical claims made in Theorem 1. We compare the NTK matrices of full fine-tuning and Propulsion fine-tuning across four different datasets: SST-2, RTE, CoLA, and STSB. The results, visualized in Figure 6, show that the NTK for Propulsion approximates the NTK for full fine-tuning with high accuracy across all datasets.

For each dataset, we compute the NTK matrices using both full fine-tuning and Propulsion fine-tuning. Specifically, the first NTK matrix, denoted as $\mathbf{K}^F(\mathbf{x},\mathbf{x}')$, corresponds to the NTK computed from fully fine-tuned models. The second NTK matrix, denoted $\mathbf{K}^P(\boldsymbol{\theta}_0\mathbf{x},\boldsymbol{\theta}_0\mathbf{x}')$, corresponds to the NTK obtained from the Propulsion method, where the base matrix $\boldsymbol{\theta}_0$ remains frozen, and only the task-specific diagonal matrix $\mathbf{Z}$ is updated. Finally, to quantify the difference between these two NTK matrices, we compute the absolute distance between them, denoted as $|\mathbf{K}^F(\mathbf{x},\mathbf{x}') - \mathbf{K}^P(\boldsymbol{\theta}_0\mathbf{x},\boldsymbol{\theta}_0\mathbf{x}')|$. This measures how

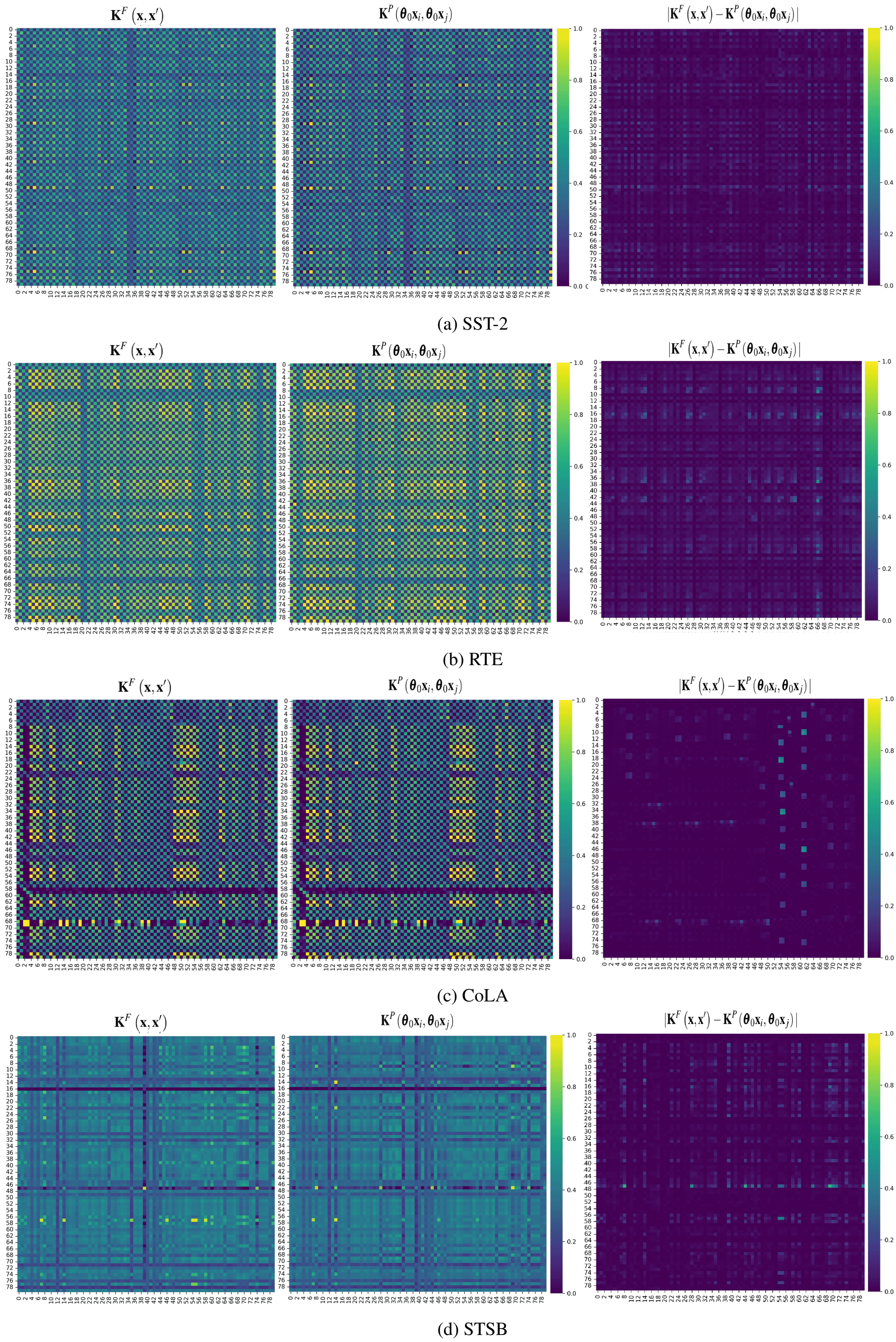


Figure 6: Heat map of NTK matrix on the SST-2, RTE, CoLA, and STSB datasets. For every dataset, the first NTK matrix is from full fine-tuning. The second NTK matrix is from the Propulsion method. The third matrix shows the absolute distance between them.

closely the NTK of Propulsion approximates the NTK of full fine-tuning.

Figure 6 presents the heatmaps of the NTK matrices across the SST-2, RTE, CoLA, and STSB datasets. The heatmaps are organized as follows: The first column corresponds to $\mathbf{K}^F(\mathbf{x}, \mathbf{x}')$, the NTK matrix computed from full fine-tuning. The second column corresponds to $\mathbf{K}^P(\boldsymbol{\theta}_0\mathbf{x}, \boldsymbol{\theta}_0\mathbf{x}')$, the NTK matrix computed from Propulsion fine-tuning. The third column shows $|\mathbf{K}^F(\mathbf{x}, \mathbf{x}') - \mathbf{K}^P(\boldsymbol{\theta}_0\mathbf{x}, \boldsymbol{\theta}_0\mathbf{x}')|$, the absolute difference between the two NTK matrices. The heatmaps demonstrate that the NTK matrices for Propulsion fine-tuning closely resemble those for full fine-tuning across all four datasets. The third column, which shows the absolute difference, indicates that the discrepancies between the two methods are minimal. This supports our theoretical findings in Section 1, which claim that Propulsion approximates full fine-tuning under the NTK regime with high probability.

These empirical results validate the claim that Propulsion, despite updating only a diagonal matrix $\mathbf{Z}$, can closely approximate the behavior of full fine-tuning in the NTK regime. This is particularly significant given that Propulsion fine-tunes a far smaller number of parameters than full fine-tuning, leading to more efficient training while maintaining comparable performance. The minimal difference observed in the third column of Figure 6 confirms that the theoretical bound on the NTK difference, as stated in Theorem 1, holds in practice.

C Kernel Behavior in the NTK Regime

In this section, we provide empirical validation of the kernel behavior in the NTK regime. As the width of the neural network tends to infinity, the gradient of the network's output with respect to its parameters stabilizes, and the network exhibits linear behavior in the parameter space. This property of NTK is crucial for understanding the training dynamics of neural networks, particularly in fine-tuning scenarios such as Propulsion.

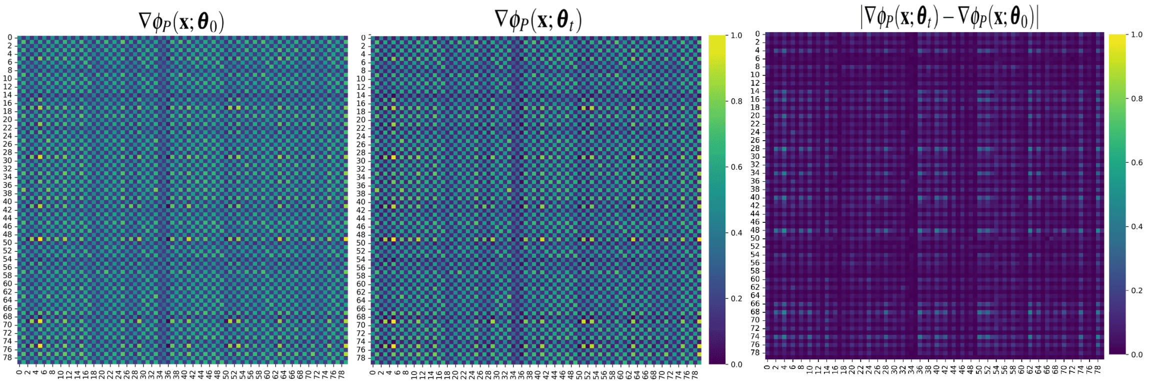
We evaluate the kernel behavior by analyzing the Jacobian matrix of the network's output with respect to the parameters $\boldsymbol{\theta}_0$ before and after several steps of training. Specifically, we compute the gradient of the model output $\phi_{\boldsymbol{\theta}}(\mathbf{x})$ with respect to the initial parameters $\boldsymbol{\theta}_0$, and compare it to the gradient after t steps of training, denoted by $\boldsymbol{\theta}_t$. For each dataset, the Jacobian matrices are computed as $\nabla_{\boldsymbol{\theta}_0}\phi(\mathbf{x})$ (the initial Jacobian matrix) and $\nabla_{\boldsymbol{\theta}_t}\phi(\mathbf{x})$ (the Jacobian matrix after t steps of training). To quantify the change in the gradients, we compute the absolute difference between the two Jacobian matrices, $\left|\nabla_{\boldsymbol{\theta}_t}\phi(\mathbf{x}) - \nabla_{\boldsymbol{\theta}_0}\phi(\mathbf{x})\right|$, which measures the stability of the gradients in the NTK regime and indicates whether the network remains in the kernel regime as training progresses.

Figure 7 presents the heatmaps of the Jacobian matrices across the SST-2, RTE, CoLA, and STSB datasets. Each row corresponds to one of the datasets and includes three columns: the first column shows $\nabla_{\boldsymbol{\theta}_0}\phi(\mathbf{x})$, the Jacobian matrix computed from the initial model parameters before training. The second column shows $\nabla_{\boldsymbol{\theta}_t}\phi(\mathbf{x})$, the Jacobian matrix computed after t steps of training. The third column shows the absolute difference between the two Jacobian matrices, $\left|\nabla_{\boldsymbol{\theta}_t}\phi(\mathbf{x}) - \nabla_{\boldsymbol{\theta}_0}\phi(\mathbf{x})\right|$. The heatmaps demonstrate that the Jacobian matrices remain relatively stable after t steps of training across all datasets. This suggests that the gradients are largely unchanged, confirming that the network is operating in the NTK regime, where the parameters exhibit kernel behavior, and the network's output becomes a linear function of the parameters.

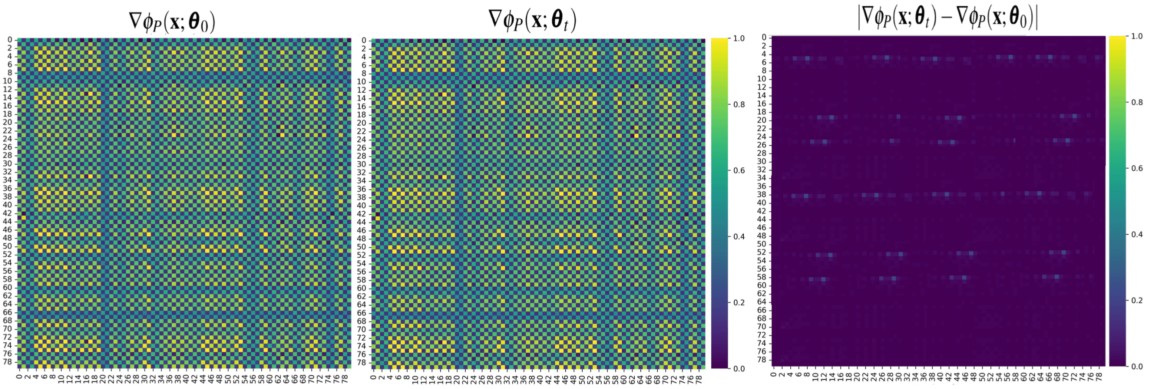
The kernel behavior observed in the Jacobian matrices across different datasets aligns with the theoretical understanding of the NTK regime. In this regime, the network's output becomes a function of the NTK matrix, and the gradients with respect to the parameters stabilize as the width of the network increases. The results in Figure 7 provide empirical evidence that, even after several steps of training, the gradients remain close to their initial values, indicating that the network has not deviated significantly from the kernel regime. This behavior is particularly relevant for fine-tuning methods like Propulsion, where the stability of gradients ensures that the model can be fine-tuned efficiently without large deviations from the pre-trained parameters. The minimal differences observed in the third column of the heatmaps confirm that the kernel behavior holds in practice, and the network remains in the NTK regime as training progresses.

D Comparison of Delta Weight Reparameterization in PEFT Methods

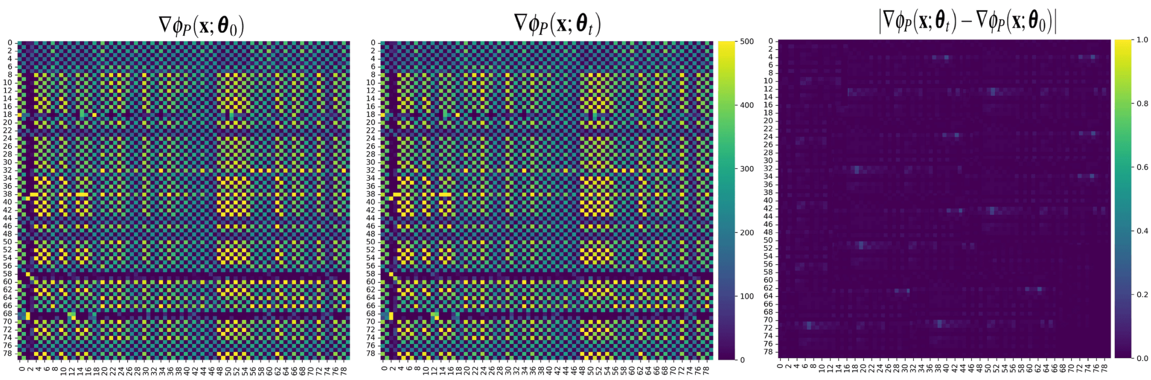
Table 6 provides a comprehensive comparison of various PEFT methods based on their reparameterization of the delta weight matrix ΔW. Each method uses different strategies for adjusting the



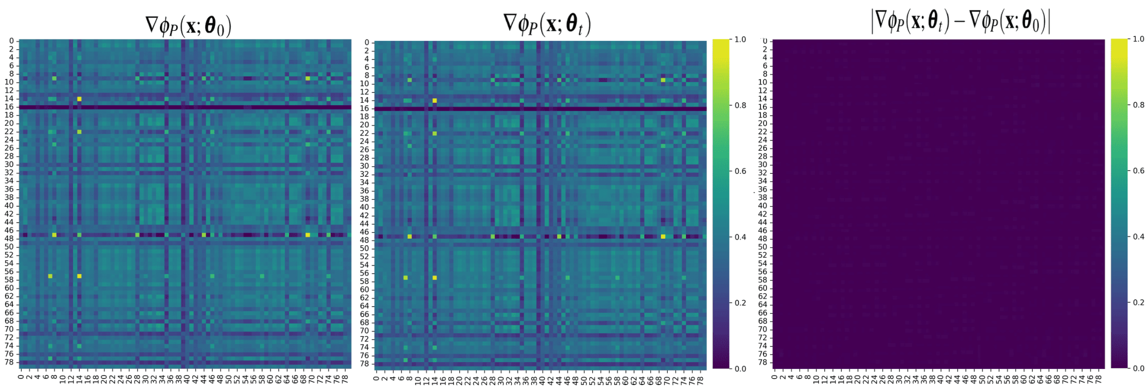
(a) SST-2



(b) RTE



(c) CoLA



(d) STSB

Figure 7: Heat map of Jacobian matrix on the SST-2, RTE, CoLA, and STSB datasets. For every dataset, the first Jacobian matrix is from the initial steps before training. The second Jacobian matrix is from the t-steps of training. The third matrix shows the absolute distance between them.

Method	ΔW Reparameterization	Notes
Intrinsic SAID	$\Delta W = F(W^r)$	$\mathrm{F}: \mathbb{R}^r \to \mathbb{R}^d$, $W^r \in \mathbb{R}^r$ are parameters to be optimized, and $r \ll d$.
LoRA	$\Delta W = W_{\text{down}} W_{\text{up}}$	$W_{\text{down}} \in \mathbb{R}^{d \times r}$, $W_{\text{up}} \in \mathbb{R}^{r \times d}$, and $r \ll \{k, d\}$.
KronA	$\Delta W = W_{\text{down}} \otimes W_{\text{up}}$	rank($W_{\text{down}} \otimes W_{\text{up}}$) = rank($W_{\text{down}}$) $\times$ rank(W_{up}).
DyLoRA	$\Delta W = W_{\text{down}\downarrow b} W_{\text{up}\downarrow b}$	$W_{\text{down}\downarrow b} = W_{\text{down}}[: b, :]$, $W_{\text{up}\downarrow b} = W_{\text{up}}[:, : b]$, $b \in \{r_{\min}, \ldots, r_{\max}\}$.
AdaLoRA	$\Delta W = PAQ$	$PP^\top = P^\top P \neq I = QQ^\top = Q^\top Q$, $\Lambda = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r)$.
IncreLoRA	$\Delta W = W_{\text{down}} \Lambda W_{\text{up}}$	$\Lambda = [\lambda_1, \lambda_2, \ldots, \lambda_r]$ with λ_i being an arbitrary constant.
DeltaLoRA	$\Delta W = W_{\text{down}} W_{\text{up}}$	$W^{(t+1)} \leftarrow W^{(t)} + \big(W_{\text{down}}^{(t+1)} W_{\text{up}}^{(t+1)} - W_{\text{down}}^{(t)} W_{\text{up}}^{(t)}\big)$.
LoRAPrune	$\Delta W = W_{\text{down}} W_{\text{up}} \odot M$	$\delta = (W + W_{\text{down}} W_{\text{up}}) \odot M$, $M \in \{0,1\}^{1 \times G}$, G is group number.
QLoRA	$\Delta W = W_{\text{down}}^{BF16} W_{\text{up}}^{BF16}$	$Y^{BF16} = X^{BF16} \cdot \mathrm{doubleDequant}(c_1^{FP32}, c_2^{FP8}, W^{NF4}) + X^{BF16} W_{\text{down}}^{BF16} W_{\text{up}}^{BF16}$.
QA-LoRA	$\Delta W = W_{\text{down}} W_{\text{up}}$	$W_{\text{down}} \in \mathbb{R}^{d \times r}$, $W_{\text{up}} \in \mathbb{R}^{r \times L}$, L is the quantization group number of W.
LoFTQ	$\Delta W = SVD(W - Q_t)$	$Q_t = q_N\big(W - W_{\text{down}}^{t-1} W_{\text{up}}^{t-1}\big)$, q_N is N-bit quantization function.
Kernel-mix	$\Delta W^h = \big(B_{\text{LoRA}}, B^h\big) \begin{pmatrix} A_{\text{LoRA}}^h \\ A^h \end{pmatrix}$	B_{LoRA} is shared across all heads, B^h, A^h provide rank-r update in each head..
LoRA-FA	$\Delta W = W_{\text{down}} W_{\text{up}} = QRW_{\text{up}}$	W_{down} is frozen, and only W_{up} is updated.
Propulsion	$\Delta W = W \odot Z$	W is frozen, and only Z is updated.

Table 6: Comparison of delta weight reparameterization across various PEFT methods. Representations of the baseline methods are taken from Xu et al. (2023).

weight updates during fine-tuning, optimizing parameter efficiency while maintaining performance.

For example, methods like LoRA and KronA employ low-rank decompositions, while methods like Propulsion, introduced in this work, use element-wise updates, where the base weights W remain frozen and only the task-specific matrix Z is updated. This comparison highlights the diverse approaches used across methods, showing how the trade-off between memory efficiency and computational complexity is handled.

E Training

Algorithm 1 describes the training process of the *Propulsion* method. We begin with an input x and a pre-trained language model $\mathbb{M}(.)$ consisting of L layers, where all parameters of $\mathbb{M}(.)$ are frozen. The *Propulsion* parameters $\mathscr{Z}$ are initialized at the beginning of training. During each training epoch, the output V is extracted from a given layer L_i. Output V is then updated to V' through element-wise multiplication with $\mathbf{z_i}^k$. This new transformed output of a given layer L_i is then sent through the rest of the model, where it is used as the input x for the subsequent layer L_{i+1}, where i ranges from 1 to N. After processing the input through all layers, the loss specific to the task is calculated, and the *Propulsion* parameters $\mathscr{Z}$ are updated based on this loss.

After we employ the *Propulsion* method to modify the outputs at all layers and fine-tune the model, we calculate the loss. We update only the *Propulsion* parameters $\mathscr{Z}$, based on the task-specific loss - the other parameters within the model remain frozen. For STS-B dataset, we have used Mean Squared Error and rest of all experiments in this study, we utilize cross-entropy loss as our objective function, which is defined below:

Algorithm 1 Propulsion PEFT training

Require: input x, a retrained LM model $\mathbb{M}(.)$ with L layers
Ensure: Freeze all parameters of $\mathbb{M}(.)$
Ensure: Initialization Propulsion parameters $\mathscr{Z}$
  while $epoch < epochs$ **do**
    for $i \leftarrow 1$ to N **do**
      $V = L_i(x)$ ▷ Output of layer L_i
      $V' = [\mathbf{v_j} \odot \mathbf{z}_i^k]_{j=1}^s$ ▷ Updating output
      $x \leftarrow V'$
    end for
    Calculating loss for task specific goal
    update parameters $\mathscr{Z}$
  end while

$$\mathscr{L}(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{T} \sum_{t=1}^{T} \mathbf{y}_t \log(\hat{\mathbf{y}}_t) \tag{16}$$

where, T represents the total number of data samples, $\mathbf{y}$ is the ground truth, and $\hat{\mathbf{y}}$ are the predicted labels. Although we focused on Transformer-based pre-trained language models to test the *Propulsion* method, it can be applied to any pre-trained Neural Network for PEFT fine-tuning because it modifies the output of each layer, independent of the model

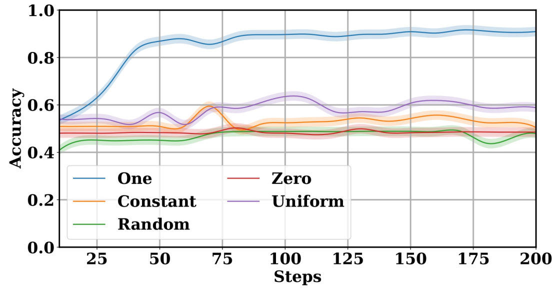


Figure 8: Performance comparison of Propulsion parameter initialization techniques

structure.

F More Ablation Study

Propulsion Parameter Initialization: Setting *Propulsion* parameters correctly is important for the model to operate accurately and efficiently. As shown in Figure 8, we tested different methods to set these parameters on the SST-2 dataset. The results clearly show that initializing the *Propulsion* parameter to 1 gives the best performance. This superior performance can be explained by the behavior of the model during the first forward pass. Specifically, when the *Propulsion* parameter is set to 1, it ensures that the output of each layer in the initial forward pass remains identical to that without any *Propulsion* modification. This approach allows the model to operate from a well-understood and predictable starting point. It uses the original output projection, which is a familiar projection of the behavior of the model, thereby facilitating smoother subsequent updates and adjustments to the *Propulsion* parameters.

Propulsion Weights After Training: In Figure 9, we observe that the *Propulsion* parameter weightings across different dimensions and layers are a crucial aspect of our analysis. Initially, the *Propulsion* weights are set to 1, and after training, they range between 0.98 and 1.02. This variation suggests that a small adjustment to the projection of the layer output is necessary to achieve a task-specific goal. The left side of the figure depicts the distribution of the *Propulsion* weights across all dimensions and layers at the start of the training, which shows uniformly set weights of 1. The right side of the figure, which focuses on a subset of dimensions, illustrates the distribution of *Propulsion* weights after training, displaying the variation in the weights. This variation indicates that the model fine-tunes the *Propulsion* parameter to optimize performance, reflecting the specific requirements of the task. These observations highlight the significance of allowing small adjustments to the *Propulsion* parameter. Even minor changes in weight can significantly impact the model's ability to meet task-specific goals. Hence, the *Propulsion* parameter plays an important role in the fine-tuning process and contributes to the overall performance of the model.

F.1 Multi-Propulsion

Instead of utilizing a single *Propulsion* vector in a layer, we can employ multiple *Propulsion* vectors to gain more control over the model's adjustments by following a pooling operation. This pooling operation dynamically synthesizes the influence of these vectors, effectively combining their effects into a single output matrix V_i'. If we use the total p numbers of *Propulsion*, then we can define the pooling operation as:

$$V_i' = \text{Pooling}(V_i^{1'}, V_i^{2'}, \ldots, V_i^{p'})$$

The pooled output V_i' is then processed as the input for the subsequent layer L_{i+1}, or can be adjusted according to specific model requirements or task-based needs.

Number of Propulsion Layers: We evaluate our model's performance on five prominent NLP benchmarks: *SST2*, *QNLI*, *MRPC*, *MNLI*, and *RTE*. As shown in Figure 10, our model maintains high accuracy across varying *Propulsion* layer counts (1 to 20.0). *SST2* achieves the highest accuracy, consistently near 95%, while *RTE* remains stable at around 80%. Across datasets, performance does not significantly fluctuate with more Propulsion layers, indicating that this method delivers robust performance across diverse tasks.

Pooling Comparison : We evaluate the impact of four pooling strategies—*Average*, *Max*, *Min*, and *L2*—on model accuracy across five benchmark datasets: *SST-2*, *MRPC*, *MNLI*, *QNLI*, and *RTE*. Figure 11 compares the different pooling methods across datasets, with *Average Pooling* consistently delivering the highest accuracy, achieving 96.83% on *SST-2* and 92.79% on *QNLI*, outperforming *Max*, *Min*, and *L2 Pooling* by up to 1.06%. On *MRPC* and *MNLI*, all pooling methods perform similarly, though *Average Pooling* maintains a slight edge. In the more challenging *RTE* dataset, differences are minimal, with *Average Pooling* at 77.64% and *L2 Pooling* at 76.83%. These results demonstrate that

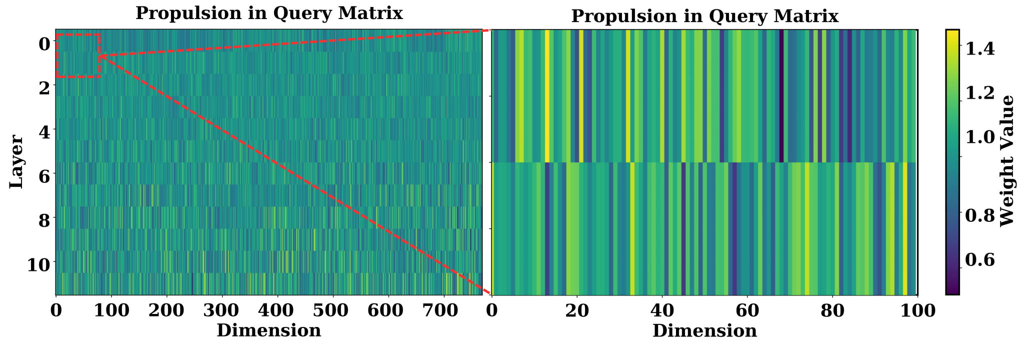


Figure 9: Visualization of trained *Propulsion* parameters across the attention query layers after fine-tuning on the SST-2 dataset. Each layer and dimension is represented, indicating the diversity of weight adjustments necessary for task-specific performance optimization.

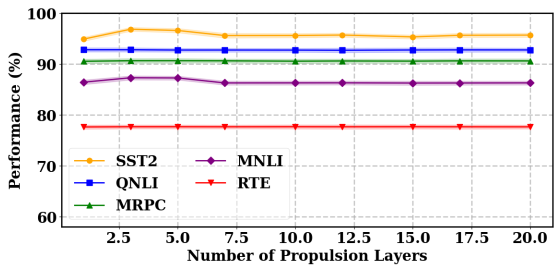


Figure 10: Model performance across five NLP benchmarks (SST2, QNLI, MRPC, MNLI, RTE) with SST2 at 95% accuracy and RTE steady at 80% across *Propulsion* units (1 to 20.0)

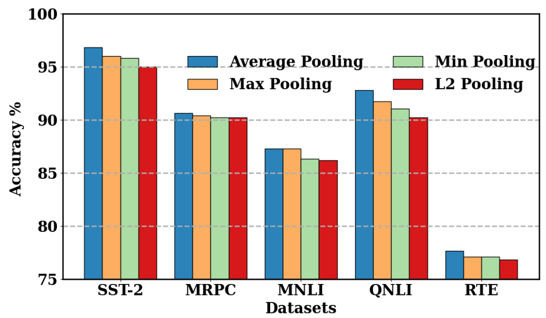


Figure 11: Accuracy comparison of pooling strategies (Average, Max, Min, L2) across five NLP datasets (SST-2, MRPC, MNLI, QNLI, RTE). Average Pooling consistently achieves the highest accuracy, while L2 Pooling tends to underperform.

Average Pooling provides the best generalization across various text classification tasks.

G Baseline Methods

Full Finetuning (FT): (Zhang et al., 2022) Full fine-tuning entails updating all pre-trained weights of a language model with task-specific data. This enables the model to learn intricate patterns, particularly specific tasks, although it requires substantial computational resources and labeled data. However, this process can result in overfitting, particularly when the task-specific dataset is limited or the model is already well suited for the target task.

Adapter$^{\mathbf{S}}$: (Houlsby et al., 2019) is a fine-tuning method that involves incorporating task-specific adapter modules into a pretrained model. This approach allows parameter-efficient tuning without requiring extensive modifications to the weights of the original model. These adapters are often characterized by their low-rank properties and include a non-linear activation function that facilitates task-specific adjustments while preserving a significant portion of the pre-trained parameters.

Prompt tuning: (Lester et al., 2021) Prompt-tuning entails appending trainable prompt tokens to the input of a language model, thereby updating only the prompt parameters through gradient descent while leaving the pretrained model's parameters frozen, which makes it a memory-efficient approach for fine-tuning. The success of prompt tuning is highly contingent upon the length and training of prompt tokens.

Prefix-tuning: (Li and Liang, 2021) Prefix-tuning is an extension of prompt tuning that introduces task-specific vectors into the activations of the multi-head attention layers of the model. These prefixes are optimized independently and do not modify the original pretrained parameters. Prefix-tuning achieves fine-tuning efficiency and stability through a parameterized feed-forward network that parameterizes prefixes.

(IA)^3: (Liu et al., 2022a) The (IA)^3 approach, which signifies Infused Adapter through Inhibiting and Amplifying Inner Activations, involves element-wise multiplication of model activations with task-specific vectors that have been learned. This strategy facilitates effective adaptation to mixed-task batches without necessitating substantial alterations to the architectural structure of the model, thereby preserving its efficiency and retaining its original form.

Bitfit: (Zaken et al., 2021) Bitfit employs a highly parameter-efficient method during fine-tuning, because it selectively updates only the bias parameters of a model. This technique capitalizes on the minimal number of parameters necessary to modify the model outputs, thereby minimizing the memory and computational resources required for full model training. **LoRA:** (Hu et al., 2021) Low-Rank Adaptation (LoRA) is a technique that fine-tunes a model by making low-rank updates to the weight matrices, enabling efficient adaptation with minimal alterations to the original parameters. This approach effectively combines the efficiency of parameter utilization and performance in subsequent tasks.

AdaLoRA: (Zhang et al., 2023) AdaLoRA is built upon LoRA and enhances its capabilities by adaptively allocating the rank and budget of updates among different weight matrices based on their importance. This approach improves both fine-tuning efficiency and task-specific performance. By dynamically adjusting the rank of the updates and concentrating on the most impactful parameters, AdaLoRA achieves a more effective outcome.

MAM Adapter: (He et al., 2021) The MAM Adapter integrates the principles of parallel adapter and prefix-tuning into a cohesive structure. Its objective is to improve model adaptation through optimized parameter allocation, and it is designed to refine various aspects of the model outputs by adjusting a combination of parameters across multiple layers.

ProPETL: (Zeng et al., 2023) These techniques are a set of hybrid fine-tuning methods that combine the aspects of adapters, prefix-tuning, and LoRA to optimize the performance across multiple tasks. By integrating multiple strategies into a cohesive approach, these methods aim to leverage the strengths of each technique, while mitigating their weaknesses.

H Evaluation Metric

In this section, we detail the evaluation metrics used to assess the performance of our models across various tasks in the GLUE benchmark suite. Each task is evaluated using specific metrics tailored to its characteristics.

For the CoLA task, we use the Matthews correlation coefficient (MCC) as the evaluation metric. MCC is particularly useful for evaluating binary classification tasks, as it considers into account true and false positives and negatives, providing a balanced measure even with imbalanced datasets.

$$\text{MCC} = \frac{\text{TP} \times \text{TN} - \text{FP} \times \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}}$$

where:

- TP = True Positives
- TN = True Negatives
- FP = False Positives
- FN = False Negatives

The MRPC and QQP tasks are both designed to assess the ability of a model to determine whether two sentences are semantically equivalent. To evaluate the performance of a model on these tasks, two metrics are used: accuracy and F1 score. Accuracy measures the percentage of correctly identified paraphrase pairs, while the F1 score provides a balance between precision and recall, offering a more nuanced view of the model's performance in identifying paraphrases.

However, the MNLI task requires the model to classify sentence pairs into one of three categories: entailment, contradiction, or neutral. To evaluate the model's performance on this task, the Average Matched Accuracy is reported, which measures the model's accuracy on the matched validation set (in-domain data). This metric reflects the model's ability to generalize across different genres, providing insights into its robustness and versatility.

Accuracy

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

F1 Score

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

where:

- $\text{Precision} = \frac{\text{TP}}{\text{TP+FP}}$
- $\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$

For the STS-B task, which involves predicting the degree of semantic similarity between sentence pairs, we use both Pearson and Spearman correlation coefficients to evaluate performance. These metrics measure the linear and rank correlations between the predicted and actual similarity scores, respectively.

Pearson Correlation

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2 \sum(y_i - \bar{y})^2}}$$

Spearman Correlation

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

where d_i is the difference between the ranks of corresponding values and n is the number of pairs.

I Dataset Description

The datasets used in this study are listed in Table 8 and 7.

Dataset	Domain	Train	Test
MultiArith	Math	–	600
AddSub	Math	–	395
GSM8K	Math	8.8K	1,319
AQuA	Math	100K	254
SingleEq	Math	–	508
SVAMP	Math	–	1,000
BoolQ	CS	9.4K	3,270
PIQA	CS	16.1K	1,830
SIQA	CS	33.4K	1,954
HellaSwag	CS	39.9K	10,042
WinoGrande	CS	63.2K	1,267
ARC-e	CS	1.1K	2,376
ARC-c	CS	2.3K	1,172
OBQA	CS	5.0K	500

Table 7: Details of datasets being evaluated. Math: arithmetic reasoning. CS: commonsense reasoning.

J Details Related Work

The development of parameter-efficient fine-tuning methods is crucial in the NLP field due to the increasing complexity of LLMs. These procedures aim to improve LM performance while reducing computational and memory requirements, as demonstrated by (Liu et al., 2022a; Nguyen et al., 2023; Chow et al., 2024). The effectiveness of PEFT techniques extends to various NLP tasks, as

Dataset	Train	Validation	Test
SQuAD v1.1	87.6k	10.6k	-
SQuAD v2.0	130k	11.9k	-
XSum	204k	11.3k	11.3k
DailyMail	287k	13.4k	11.5k
CoLA	8.55k	1.04k	1.06k
SST2	67.3k	872	1.82k
MRPC	3.67k	408	1.73k
STS-B	5.75k	1.5k	1.38k
QQP	364k	40.4k	391k
MNLI	393k	9.8k	9.8k
QNLI	105k	5.46k	5.46k
RTE	2.49k	277	3k

Table 8: Data Description of Glue, Question Answering, Text Summarizing

shown by (Fu et al., 2023; He et al., 2021). Several researchers, including Liu et al. (2021b, 2023); Zhang et al. (2023); Hu et al. (2021); Li and Liang (2021); Zaken et al. (2021) have proposed methods targeting the challenge of increasing LLM performance with reduced computational and memory demands. Studies have found these methods highly effective for NLP tasks, highlighting their potential for practical applications.

Prompt Tuning is a technique used to improve natural language understanding and generation tasks by adjusting learnable parameters (Lester et al., 2021). Researchers have added residual connections to improve performance and stability, and have extended it to continual learning (Razdaibiedina et al., 2023b,a). Recent studies have explored real-time transformation with dynamic prompt tuning (Yang et al., 2023b) and multilevel control (Wang et al., 2022) through hierarchical prompt tuning. Additionally, multimodal prompt tuning has been developed to integrate multiple data types and improve model performance. Techniques such as MixPrompt (Yang et al., 2023a) and E2VPT (Han et al., 2023) have been employed to combine input and key-value prompts, while prefix-tuning (Li and Liang, 2021) has been used to add learnable parameters to a pre-trained model's input for various NLP tasks. Hierarchical prefix-tuning has been implemented to provide better control over model behavior (Chen et al., 2022a) , and dynamic prefix-tuning has been developed for real-time adaptation based on context (Liu et al., 2022b).

Low-Rank Adaptation (LoRA) is a memory-efficient method for fine tuning pre-trained models that was introduced in a study conducted by Hu et al. (2021). In subsequent research, Renduchintala et al. (2023); Sheng et al. (2023); Xia et al. (2024) proposed extensions for multitask learning that were applied to practical scenarios by Wang et al. (2023). In addition, Dettmers et al. (2024) investigated memory optimization. Lialin et al. (2023) introduced ReLoRA, a variant designed for Pre-training that requires a full-rank warm-up phase. Notable contributions in this field include (Zhang et al., 2023), which dynamically adjusts low-rank adaptation during training, and the Low-Rank Kronecker Product (LoKr) proposed by Edalati et al. (2022), which focuses on knowledge retention across tasks. ResLoRA, by Shi et al. (2024), includes the use of residual paths during the training and merging techniques to eliminate these paths during the inference process. Finally, Hyeon-Woo et al. (2021) introduced the Low-Rank Hadamard Product (LoHa), that utilizes hierarchical adaptation strategies.

Subspace learning focuses on the learning processes that can be successfully conducted within a lower-dimensional parameter space (Larsen et al., 2021; Gur-Ari et al., 2018). This approach involves optimizing model weights within a low-rank subspace and has been widely implemented in various machine-learning domains, including meta-learning and continual learning (Lee and Choi, 2018; Chaudhry et al., 2020). Recent advancements have investigated the potential of subspace learning to improve the model generalization and robustness. For instance, Nunez et al. (2023) introduced adaptive subspace learning methods that dynamically adjust the subspace during training, resulting in an improved performance across various tasks. Furthermore, the integration of subspace learning with neural architecture search has shown promising results in identifying efficient model architectures (Chen et al., 2022b).

Projected Gradient Descent (PGD) has been improved by the GaLore method, which specifically targets gradient shapes in multilayer neural networks rather than treating the objective function as an arbitrary nonlinear black-box function Zhao et al. (2024); Chen and Wainwright (2015); Chen et al. (2019). Recent research has emphasized the effectiveness of the GaLore method Zhao et al. (2024) in addressing the intricacies of neural network training, making it a valuable tool for optimizing training procedures. Moreover, additional research has indicated that GaLore presents a benefit in obtaining more rapid convergence rates and stability for high-dimensional datasets Zhang and Fan (2024). Recent developments comprise of methods for addressing sparsity and redundancy in neural network gradients, which contribute to increasing training efficiency Zhao et al. (2024), representing a substantial advancement in neural network optimization.

Memory-efficient optimization a vital aspect of adaptive optimization algorithms , aims to decrease memory requirements. Studies such as those conducted by Shazeer and Stern (2018) emphasize the importance of this principle. In addition, quantization methods were employed to decrease the memory costs of the optimizer state, and a fused gradient computation was proposed to minimize the weight gradient memory during training Li et al. (2024). Furthermore, recent advancements include hierarchical memory management for dynamic memory allocation during training and sparse gradient updates to selectively reduce memory usage (Li et al., 2024).

K LLM Performance

K.1 Sequence Classification

In the field of classification, we conducted a comprehensive evaluation of various LLMs, including Bloom (Le Scao et al., 2023), Llama2 (Touvron et al., 2023), Falcon (Almazrouei et al., 2023), Mistral (Jiang et al., 2023), and Phi-2 (Ranjit et al., 2024), employing different fine-tuning techniques. For each model, we examined the effectiveness of traditional approaches such as Finetuning, Prefix-Tuning, Prompt Tuning, PTuning, LoRA Rank 1, and LoRA Rank 2, and compared them to our proposed *Propulsion* methods, both for Propulsion(All) and Propulsion(Attn). Notably, *Propulsion* consistently outperforms traditional methods across different datasets, showcasing its superior efficiency and effectiveness. The performances of various models on different datasets are documented in Table 9, 10, 11, 12, and 13.

Across the "Fake News Filipino" dataset, *Propulsion*, especially when applied as Propulsion(All), demonstrates remarkable performance improvements compared to traditional approaches. It achieves the highest accuracy and F1-score, emphasizing its capability to efficiently adapt LLMs to specific tasks while minimizing trainable parameters. In the "Emotion" dataset, *Propulsion* consistently outperforms other methods, indicating its robustness across different classification tasks. The same trend is observed in the "SST-2" dataset, where *Propulsion* invariably achieves superior results. Lastly, in the "Cola" dataset, Propulsion(All) and Propulsion(Attn) perpetually outperform other approaches, underscoring their potential for enhancing sequence classification tasks.

Comparatively, traditional methods like Propulsion(All) and Propulsion(Attn), although efficient in terms of parameters compared to fine-tuning, tend to lag behind *Propulsion* in terms of accuracy and F1-score. Furthermore, *Propulsion* requires fewer trainable parameters, making it an attractive choice for practitioners aiming to optimize performance while maintaining efficiency.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
Fake News Filipino	Full Fine-tuning	100.000	95.02	93.83
	Prefix-Tuning	0.03493	70.99	68.18
	Prompt Tuning	0.00701	74.31	72.23
	P-Tuning	0.01582	72.97	70.19
	LoRA Rank 1	0.01413	90.13	88.87
	LoRA Rank 2	0.05794	**93.56**	90.05
	Propulsion(All)	0.00032	92.98	**90.75**
	Propulsion(Attn)	**0.00014**	91.14	89.26
Emotion	Full Fine-tuning	100.000	90.31	87.52
	Prefix-Tuning	0.03521	74.75	68.11
	Prompt Tuning	0.00813	79.12	71.07
	P-Tuning	0.01593	69.45	70.23
	LoRA Rank 1	0.02413	86.76	80.23
	LoRA Rank 2	0.06831	87.52	82.01
	Propulsion(All)	0.00159	**88.32**	**82.75**
	Propulsion(Attn)	**0.00102**	86.93	82.26
SST2	Full Fine-tuning	100.000	97.93	97.81
	Prefix-Tuning	0.03493	85.78	86.31
	Prompt Tuning	0.00715	92.45	92.78
	P-Tuning	0.01653	91.34	91.75
	LoRA Rank 1	0.01456	92.27	92.77
	LoRA Rank 2	0.02831	94.36	94.83
	Propulsion(All)	0.00080	**96.95**	**96.74**
	Propulsion(Attn)	**0.00031**	96.64	96.27
Cola	Full Fine-tuning	100.000	87.05	89.93
	Prefix-Tuning	0.03495	73.72	83.69
	Prompt Tuning	0.00723	82.74	**87.70**
	P-Tuning	0.01615	70.32	81.12
	LoRA Rank 1	0.01415	81.13	83.03
	LoRA Rank 2	0.02797	84.33	85.21
	Propulsion(All)	0.00079	**84.99**	86.22
	Propulsion(Attn)	**0.00048**	84.62	85.98

Table 9: Sequence Classification Results for the Bloom Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity.

K.2 Token Classification

Tables 14, 15, 16, 17, and 18 compare the results of *Propulsion* and other PEFT methods on token classification. The majority of experiments on token classification show *Propulsion* having higher accuracy and F1-scores compared to the other PEFT methods tested. The accuracy under *Propulsion* is still less than full fine-tuning, but remains higher amongst the other PEFT methods.

Amongst the two Propulsion applications, there seems to be a mix as to which *Propulsion* method provides the best improvement. Within the conl103

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
Fake News Filipino	Full Fine-tuning	100.000	95.22	93.90
	Prefix-Tuning	0.03983	70.06	68.57
	Prompt Tuning	0.00743	73.72	72.07
	P-Tuning	0.01731	71.54	70.63
	LoRA Rank 1	0.01601	90.38	87.62
	LoRA Rank 2	0.03213	92.14	**90.86**
	Propulsion(All)	**0.00021**	**92.37**	89.98
	Propulsion(Attn)	0.00032	90.95	88.32
Emotion	Full Fine-tuning	100.000	91.11	87.92
	Prefix-Tuning	0.03994	84.31	82.78
	Prompt Tuning	0.00864	85.37	82.50
	P-Tuning	0.01781	83.05	81.88
	LoRA Rank 1	0.01624	86.49	82.86
	LoRA Rank 2	0.03233	88.56	**84.18**
	Propulsion(All)	0.00171	**88.82**	83.63
	Propulsion(Attn)	**0.00120**	85.97	82.91
SST2	Full Fine-tuning	100.000	97.32	97.69
	Prefix-Tuning	0.04855	85.78	86.31
	Prompt Tuning	0.00712	94.24	**97.26**
	P-Tuning	0.01753	95.55	96.62
	LoRA Rank 1	0.01607	86.97	81.93
	LoRA Rank 2	0.03191	87.11	82.03
	Propulsion(All)	0.00083	**96.62**	96.56
	Propulsion(Attn)	**0.00034**	96.60	96.45
Cola	Full Fine-tuning	100.000	88.22	89.64
	Prefix-Tuning	0.03984	71.18	83.29
	Prompt Tuning	0.00757	73.27	85.26
	P-Tuning	0.01751	69.12	81.74
	LoRA Rank 1	0.01603	82.25	83.43
	LoRA Rank 2	0.03213	84.18	83.88
	Propulsion(All)	0.00090	**85.21**	**86.33**
	Propulsion(Attn)	**0.00058**	84.46	85.95

Table 10: Sequence Classification Results for the Llama2 Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
Fake News Filipino	Full Fine-tuning	100.000	94.05	92.93
	Prefix-Tuning	0.03821	69.57	68.19
	Prompt Tuning	0.00732	72.35	70.78
	P-Tuning	0.01797	70.23	69.15
	LoRA Rank 1	0.00972	88.31	85.14
	LoRA Rank 2	0.05784	**91.89**	**89.44**
	Propulsion(All)	0.00072	90.21	88.97
	Propulsion(Attn)	**0.00027**	90.32	87.35
Emotion	Full Fine-tuning	100.000	88.53	85.94
	Prefix-Tuning	0.03836	81.14	80.61
	Prompt Tuning	0.00841	87.25	84.19
	P-Tuning	0.01803	81.76	79.14
	LoRA Rank 1	0.01194	84.17	82.34
	LoRA Rank 2	0.05781	**88.79**	**86.13**
	Propulsion(All)	0.00201	87.01	82.22
	Propulsion(Attn)	**0.00111**	86.39	82.14
SST2	Full Fine-tuning	100.000	96.23	95.76
	Prefix-Tuning	0.03818	90.18	91.36
	Prompt Tuning	0.00605	93.56	93.75
	P-Tuning	0.01781	90.33	91.26
	LoRA Rank 1	0.01193	91.13	92.07
	LoRA Rank 2	0.05789	91.72	92.17
	Propulsion(All)	0.00090	94.83	94.21
	Propulsion(Attn)	**0.00035**	**95.23**	**95.18**
Cola	Full Fine-tuning	100.000	85.22	87.39
	Prefix-Tuning	0.03826	70.03	82.23
	Prompt Tuning	0.00711	71.45	84.47
	P-Tuning	0.01792	68.07	81.73
	LoRA Rank 1	0.00973	82.14	82.38
	LoRA Rank 2	0.05741	**84.66**	**85.33**
	Propulsion(All)	0.00091	83.84	85.13
	Propulsion(Attn)	**0.00062**	84.21	85.33

Table 11: Sequence Classification Results for the Falcon Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

Dataset	Type	Parameters	Accuracy (%)	F1-score (%)
Fake News Filipino	Full Fine-tuning	100.000	97.92	94.72
	Prefix-Tuning	0.03651	71.26	70.91
	Prompt Tuning	0.00169	74.12	72.27
	P-Tuning	0.01753	71.37	71.95
	LoRA Rank 1	0.07502	91.28	90.05
	LoRA Rank 2	0.17129	92.19	89.18
	Propulsion(All)	0.00017	**94.15**	**91.96**
	Propulsion(Attn)	**0.00024**	92.54	90.16
Emotion	Full Fine-tuning	100.000	93.53	89.09
	Prefix-Tuning	0.03683	82.19	79.24
	Prompt Tuning	0.00736	86.17	81.77
	P-Tuning	0.01783	83.14	80.01
	LoRA Rank 1	0.01539	84.37	80.08
	LoRA Rank 2	0.01731	88.45	84.23
	Propulsion(All)	0.00160	88.83	82.61
	Propulsion(Attn)	**0.00112**	**89.23**	**84.99**
SST2	Full Fine-tuning	100.000	98.09	98.98
	Prefix-Tuning	0.03673	91.20	92.28
	Prompt Tuning	0.00618	93.14	93.47
	P-Tuning	0.01764	90.76	91.15
	LoRA Rank 1	0.01512	92.65	93.03
	LoRA Rank 2	0.01726	94.53	94.67
	Propulsion(All)	0.00078	**97.01**	96.07
	Propulsion(Attn)	**0.00029**	96.82	**97.25**
Cola	Full Fine-tuning	100.000	87.75	89.90
	Prefix-Tuning	0.03652	72.21	80.43
	Prompt Tuning	0.00639	74.13	81.66
	P-Tuning	0.01754	71.23	79.76
	LoRA Rank 1	0.01505	83.44	84.65
	LoRA Rank 2	0.01712	85.32	86.04
	Propulsion(All)	0.00080	**86.07**	86.32
	Propulsion(Attn)	**0.00042**	85.01	**86.36**

Table 12: Sequence Classification Results for the Mistral Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
Fake News Filipino	Full Fine-tuning	100.000	92.43	90.71
	Prefix-Tuning	0.83914	66.28	66.31
	Prompt Tuning	0.14124	68.15	67.22
	P-Tuning	0.15824	67.33	66.87
	LoRA Rank 1	0.13741	83.35	81.46
	LoRA Rank 2	0.71651	86.67	84.29
	Propulsion(All)	**0.04261**	**89.46**	**88.73**
	Propulsion(Attn)	0.04921	88.74	87.21
Emotion	Full Fine-tuning	100.000	87.95	84.78
	Prefix-Tuning	0.86523	77.27	76.25
	Prompt Tuning	0.14234	82.16	80.43
	P-Tuning	0.15845	77.36	75.81
	LoRA Rank 1	0.13748	82.67	80.25
	LoRA Rank 2	0.71656	85.03	82.66
	Propulsion(All)	0.06269	82.72	80.71
	Propulsion(Attn)	**0.02419**	**85.94**	**83.24**
SST2	Full Fine-tuning	100.000	94.63	94.24
	Prefix-Tuning	0.83721	86.24	87.13
	Prompt Tuning	0.14231	88.19	88.04
	P-Tuning	0.15851	85.43	87.68
	LoRA Rank 1	0.13753	86.21	87.18
	LoRA Rank 2	0.71668	86.75	88.28
	Propulsion(All)	0.02740	**96.95**	**96.75**
	Propulsion(Attn)	**0.01470**	96.63	96.29
Cola	Full Fine-tuning	100.000	84.23	85.13
	Prefix-Tuning	0.82621	66.24	70.16
	Prompt Tuning	0.14123	67.47	70.81
	P-Tuning	0.15833	64.36	68.38
	LoRA Rank 1	0.13744	78.55	80.26
	LoRA Rank 2	0.71654	80.39	82.43
	Propulsion(All)	**0.03671**	80.97	82.21
	Propulsion(Attn)	0.05140	**81.41**	**82.74**

Table 13: Sequence Classification Results for the Phi-2 Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

dataset, Propulsion(Attn) provided the highest accuracy and F1-scores on four of the five LLMs tested. In contrast, Propulsion(All) had higher accuracy and F1-scores than Propulsion(Attn) on the WikiAnn dataset. This may indicate that the layers *Propulsion* may depend on the use case. Regardless of dataset, however, *Propulsion* applied to any combination of layers showed either similar or improved metrics while significantly reducing parameter size.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
conll03	Full Fine-tuning	100.000	98.53	82.47
	Prefix-Tuning	0.03534	83.55	24.86
	Prompt Tuning	0.00843	85.23	28.73
	P-Tuning	0.01583	83.22	26.34
	LoRA Rank 1	0.01403	91.12	68.24
	LoRA Rank 2	0.06795	93.23	71.33
	Propulsion(All)	0.00068	94.18	71.69
	Propulsion(Attn)	**0.00049**	**94.21**	**71.70**
NCBI disease	Full Fine-tuning	100.000	98.53	92.46
	Prefix-Tuning	0.03492	89.09	60.06
	Prompt Tuning	0.00742	91.17	75.34
	P-Tuning	0.01572	90.22	81.23
	LoRA Rank 1	0.01417	92.86	80.00
	LoRA Rank 2	0.06797	96.12	83.49
	Propulsion(All)	0.00091	**96.27**	**84.95**
	Propulsion(Attn)	**0.00066**	95.42	82.28
WikiAnn	Full Fine-tuning	100.000	90.50	60.14
	Prefix-Tuning	0.03527	71.67	22.18
	Prompt Tuning	0.00732	76.23	31.78
	P-Tuning	0.01577	70.65	24.33
	LoRA Rank 1	0.01408	82.23	41.23
	LoRA Rank 2	0.06791	**85.13**	**45.14**
	Propulsion(All)	0.00081	83.29	42.23
	Propulsion(Attn)	**0.00042**	82.69	42.21

Table 14: Token Classification Results for the Bloom Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
conll03	Full Fine-tuning	100.000	98.75	80.77
	Prefix-Tuning	0.03964	82.28	66.56
	Prompt Tuning	0.00638	86.65	69.91
	P-Tuning	0.01731	80.11	65.11
	LoRA Rank 1	0.01426	88.67	63.34
	LoRA Rank 2	0.07122	91.32	69.03
	Propulsion(All)	**0.00040**	**93.73**	**70.93**
	Propulsion(Attn)	0.00069	93.12	70.29
NCBI disease	Full Fine-tuning	100.000	98.32	93.38
	Prefix-Tuning	0.03976	88.23	68.23
	Prompt Tuning	0.00712	91.22	78.24
	P-Tuning	0.01733	90.15	77.23
	LoRA Rank 1	0.01424	92.48	80.18
	LoRA Rank 2	0.07125	95.34	82.87
	Propulsion(All)	0.00081	96.33	84.84
	Propulsion(Attn)	**0.00060**	**96.28**	**84.89**
WikiAnn	Full Fine-tuning	100.000	91.49	63.21
	Prefix-Tuning	0.03986	81.15	35.17
	Prompt Tuning	0.00712	83.23	44.19
	P-Tuning	0.01743	81.29	38.11
	LoRA Rank 1	0.01434	84.82	47.90
	LoRA Rank 2	0.07125	86.56	49.39
	Propulsion(All)	0.00079	**86.89**	**50.71**
	Propulsion(Attn)	**0.00048**	86.79	49.64

Table 15: Token Classification Results for the Llama2 Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
conll03	Full Fine-tuning	100.000	97.82	79.03
	Prefix-Tuning	0.03772	90.57	67.62
	Prompt Tuning	0.00832	91.26	70.15
	P-Tuning	0.01762	89.23	66.02
	LoRA Rank 1	0.01942	90.21	68.96
	LoRA Rank 2	0.09752	93.25	71.19
	Propulsion(All)	0.00068	94.31	71.83
	Propulsion(Attn)	**0.00051**	**94.87**	**72.08**
NCBI disease	Full Fine-tuning	100.000	97.93	90.88
	Prefix-Tuning	0.03763	89.23	69.33
	Prompt Tuning	0.00721	92.05	82.28
	P-Tuning	0.01752	88.15	70.36
	LoRA Rank 1	0.01936	90.55	80.25
	LoRA Rank 2	0.09754	94.41	83.19
	Propulsion(All)	0.00082	95.73	82.08
	Propulsion(Attn)	**0.00053**	**96.12**	**84.38**
WikiAnn	Full Fine-tuning	100.000	89.23	62.09
	Prefix-Tuning	0.03772	82.67	36.55
	Prompt Tuning	0.00836	83.33	43.32
	P-Tuning	0.01768	81.14	35.21
	LoRA Rank 1	0.01983	80.47	41.58
	LoRA Rank 2	0.09752	86.61	**48.03**
	Propulsion(All)	0.00060	**82.89**	42.61
	Propulsion(Attn))	**0.00041**	82.86	42.39

Table 16: Token Classification Results for the Falcon Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
conll03	Full Fine-tuning	100.000	98.89	84.60
	Prefix-Tuning	0.03634	83.31	58.54
	Prompt Tuning	0.00741	87.77	62.19
	P-Tuning	0.01743	81.15	67.59
	LoRA Rank 1	0.01494	88.32	68.14
	LoRA Rank 2	0.08694	92.05	70.66
	Propulsion(All)	0.00060	95.39	72.80
	Propulsion(Attn)	**0.00040**	**95.99**	**72.13**
NCBI disease	Full Fine-tuning	100.000	98.52	93.39
	Prefix-Tuning	0.03627	88.49	74.25
	Prompt Tuning	0.00696	92.03	80.11
	P-Tuning	0.01735	87.13	63.29
	LoRA Rank 1	0.01483	94.58	82.37
	LoRA Rank 2	0.08698	96.88	83.15
	Propulsion(All)	0.00078	96.81	85.16
	Propulsion(Attn)	**0.00049**	**97.09**	**85.13**
WikiAnn	Full Fine-tuning	100.000	92.15	63.09
	Prefix-Tuning	0.03633	81.91	36.03
	Prompt Tuning	0.00752	84.48	45.31
	P-Tuning	0.01733	81.04	35.02
	LoRA Rank 1	0.01495	82.08	42.22
	LoRA Rank 2	0.08692	85.33	45.95
	Propulsion(All)	0.00090	**86.63**	**46.29**
	Propulsion(Attn)	**0.00048**	85.19	45.62

Table 17: Token Classification Results for the Mistral Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
conll03	Full Fine-tuning	100.000	98.13	79.02
	Prefix-Tuning	0.83844	78.27	56.63
	Prompt Tuning	0.14124	80.38	58.27
	P-Tuning	0.15814	76.54	56.07
	LoRA Rank 1	0.13746	84.44	62.15
	LoRA Rank 2	0.71649	86.56	65.43
	Propulsion(All)	0.00949	90.52	71.83
	Propulsion(Attn)	**0.00835**	**91.18**	**71.88**
NCBI disease	Full Fine-tuning	100.000	95.82	91.19
	Prefix-Tuning	0.83823	82.42	63.38
	Prompt Tuning	0.13939	85.61	65.44
	P-Tuning	0.15794	81.17	67.63
	LoRA Rank 1	0.13748	86.23	78.45
	LoRA Rank 2	0.71493	87.34	78.26
	Propulsion(All)	0.00990	89.32	80.74
	Propulsion(Attn)	**0.00434**	**90.93**	**81.87**
WikiAnn	Full Fine-tuning	100.000	88.92	58.21
	Prefix-Tuning	0.83832	74.37	31.57
	Prompt Tuning	0.01416	78.86	38.32
	P-Tuning	0.15812	75.23	32.26
	LoRA Rank 1	0.13748	79.04	39.88
	LoRA Rank 2	0.71649	81.53	**44.47**
	Propulsion(All)	0.00847	82.08	42.97
	Propulsion(Attn)	**0.00690**	**83.28**	43.01

Table 18: Token Classification Results for the Phi-2 Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

K.3 Entailment Detection

The results of entailment detection using various models, including Bloom, Llama2, Falcon, Mistral, and Phi-2, are presented in Tables 19, 20, 21, 22, and 23. Across all three datasets (RTE, MRPC, SNLI), full fine-tuning consistently achieves the highest accuracy and F1-score, with Bloom and Mistral models demonstrating remarkable results. This underscores the value of fine-tuning the entire model's parameters to adapt to specific entailment tasks, as it allows the model to capture intricate patterns and nuances in the data.

In contrast, Propulsion(All) and Propulsion(Attn) techniques, which involve fine-tuning only a small fraction of the model's parameters, tend to yield significantly lower accuracy and F1-scores. This suggests that limiting parameter updates to specific Propulsion(All) or Propulsion(Attn) may not be sufficient for optimal entailment classification performance, as these methods may struggle to capture the diverse and complex relationships present in the data.

The LoRA Rank 1 and LoRA Rank 2 models deliver competitive results, particularly evident in the RTE dataset, where they outperform other techniques. This indicates that techniques like LoRA Rank, which involve a moderate amount of parameter modification, can strike a balance between model adaptation and computational efficiency.

However, Propulsion, whether applied to Propulsion(All) or Propulsion(Attn), consistently performs well across datasets, demonstrating its effectiveness as an alternative fine-tuning strategy. Propulsion achieves strong results with a minimal increase in the number of parameters, making it a promising approach for entailment classification tasks where computational resources are a concern.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
RTE	Full Fine-tuning	100.000	92.31	87.19
	Prefix-Tuning	0.03493	70.03	64.06
	Prompt Tuning	0.00714	65.34	62.20
	P-Tuning	0.01584	71.11	69.23
	LoRA Rank 1	0.01402	80.25	80.01
	LoRA Rank 2	0.05804	84.45	83.26
	Propulsion(All)	0.00070	83.98	82.86
	Propulsion(Attn)	**0.00049**	**84.98**	**83.97**
MRPC	Full Fine-tuning	100.000	90.01	91.13
	Prefix-Tuning	0.03494	73.56	81.70
	Prompt Tuning	0.00773	81.39	86.01
	P-Tuning	0.01562	78.08	84.38
	LoRA Rank 1	0.01393	80.21	82.29
	LoRA Rank 2	0.05799	83.88	84.84
	Propulsion(All)	0.00080	88.99	86.28
	Propulsion(Attn)	**0.00050**	**89.13**	**86.47**
SNLI	Full Fine-tuning	100.000	95.62	95.78
	Prefix-Tuning	0.03492	87.32	87.26
	Prompt Tuning	0.00803	88.88	88.87
	P-Tuning	0.01594	86.22	86.54
	LoRA Rank 1	0.01412	91.37	91.36
	LoRA Rank 2	0.05813	93.23	93.68
	Propulsion(All)	0.0008-	92.64	92.88
	Propulsion(Attn)	**0.00056**	**93.75**	**94.02**

Table 19: Entailment Classification Results for the Bloom Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
RTE	Full Fine-tuning	100.000	93.51	88.92
	Prefix-Tuning	0.03982	70.15	65.23
	Prompt Tuning	0.00737	62.81	66.00
	P-Tuning	0.01753	67.24	66.21
	LoRA Rank 1	0.01612	81.04	80.67
	LoRA Rank 2	0.03224	83.43	81.44
	Propulsion(All)	0.00071	**85.83**	**84.12**
	Propulsion(Attn)	**0.00048**	83.82	83.53
MRPC	Full Fine-tuning	100.000	92.25	92.95
	Prefix-Tuning	0.03973	79.41	80.01
	Prompt Tuning	0.00724	80.18	80.37
	P-Tuning	0.01745	74.56	82.67
	LoRA Rank 1	0.01601	80.48	82.02
	LoRA Rank 2	0.03218	81.89	83.11
	Propulsion(All)	0.00079	**85.97**	**86.37**
	Propulsion(Attn)	**0.00047**	85.13	85.47
SNLI	Full Fine-tuning	100.000	93.31	94.03
	Prefix-Tuning	0.03986	86.34	86.33
	Prompt Tuning	0.00736	87.02	87.41
	P-Tuning	0.01752	85.17	86.27
	LoRA Rank 1	0.01613	90.21	90.87
	LoRA Rank 2	0.03228	91.15	91.85
	Propulsion(All)	0.00090	**91.53**	**91.91**
	Propulsion(Attn)	**0.00064**	90.89	91.14

Table 20: Entailment Classification Results for the Llama2 Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
RTE	Full Fine-tuning	100.000	93.22	87.67
	Prefix-Tuning	0.03822	64.23	63.38
	Prompt Tuning	0.00813	66.51	66.02
	P-Tuning	0.01794	53.42	53.09
	LoRA Rank 1	0.01138	73.28	70.15
	LoRA Rank 2	0.01774	78.33	73.42
	Propulsion(All))	0.00080	80.22	79.83
	Propulsion(Attn)	**0.00064**	**80.35**	**79.88**
MRPC	Full Fine-tuning	100.000	90.21	90.83
	Prefix-Tuning	0.03813	74.13	78.22
	Prompt Tuning	0.00715	80.04	80.19
	P-Tuning	0.01783	80.43	79.59
	LoRA Rank 1	0.00983	80.82	82.21
	LoRA Rank 2	0.01763	82.52	83.01
	Propulsion(All)	0.00072	82.78	83.60
	Propulsion(Attn)	**0.00050**	83.13	85.27
SNLI	Full Fine-tuning	100.000	92.53	92.97
	Prefix-Tuning	0.03822	84.33	84.98
	Prompt Tuning	0.00843	86.13	86.93
	P-Tuning	0.01782	83.31	83.66
	LoRA Rank 1	0.01163	87.05	87.29
	LoRA Rank 2	0.06773	89.21	89.88
	Propulsion(All)	0.00068	**90.80**	91.02
	Propulsion(Attn)	**0.00049**	**90.81**	91.03

Table 21: Entailment Classification Results for the Falcon Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
RTE	Full Fine-tuning	100.000	94.67	89.82
	Prefix-Tuning	0.03663	76.22	74.45
	Prompt Tuning	0.00732	80.34	80.17
	P-Tuning	0.01778	75.12	75.86
	LoRA Rank 1	0.01521	83.39	82.25
	LoRA Rank 2	0.06739	85.65	83.12
	Propulsion(All)	0.00080	84.83	83.77
	Propulsion(Attn)	**0.00061**	**85.84**	**84.77**
MRPC	Full Fine-tuning	100.000	93.02	94.21
	Prefix-Tuning	0.03654	75.28	77.03
	Prompt Tuning	0.00722	80.34	82.17
	P-Tuning	0.01715	76.19	80.31
	LoRA Rank 1	0.01513	82.83	83.41
	LoRA Rank 2	0.06724	**86.47**	87.02
	Propulsion(All)	0.00078	85.73	85.27
	Propulsion(Attn)	**0.00050**	86.41	**87.88**
SNLI	Full Fine-tuning	100.000	94.21	95.32
	Prefix-Tuning	0.03666	85.55	85.78
	Prompt Tuning	0.00744	86.35	86.21
	P-Tuning	0.01774	85.37	86.05
	LoRA Rank 1	0.01524	84.12	84.76
	LoRA Rank 2	0.06736	89.11	89.77
	Propulsion(All)	0.00085	91.72	91.41
	Propulsion(Attn)	**0.00063**	**92.66**	**91.80**

Table 22: Entailment Classification Results for the Mistral Model. The best results are highlighted in **bold**, and the second-best result is underlined for clarity except full fine-tuning.

Dataset	Type	Parameters (%)	Accuracy (%)	F1-score (%)
RTE	Full Fine-tuning	100.000	90.37	85.74
	Prefix-Tuning	0.83872	59.54	58.27
	Prompt Tuning	0.14234	61.18	61.84
	P-Tuning	0.15834	58.61	56.38
	LoRA Rank 1	0.13746	66.52	65.82
	LoRA Rank 2	0.71658	72.25	70.45
	Propulsion(All)	<u>0.00421</u>	<u>76.54</u>	<u>75.89</u>
	Propulsion(Attn)	**0.00250**	**76.63**	**76.21**
MRPC	Full Fine-tuning	100.000	89.31	90.21
	Prefix-Tuning	0.83822	71.15	72.78
	Prompt Tuning	0.14345	73.16	75.28
	P-Tuning	0.15842	70.48	71.21
	LoRA Rank 1	0.13747	80.53	81.33
	LoRA Rank 2	0.71659	<u>83.19</u>	<u>84.23</u>
	Propulsion(All)	<u>0.00739</u>	**83.73**	**84.82**
	Propulsion(Attn)	**0.00345**	82.64	83.52
SNLI	Full Fine-tuning	100.00	90.54	91.02
	Prefix-Tuning	0.83844	79.27	79.82
	Prompt Tuning	0.14149	81.30	81.80
	P-Tuning	0.15823	78.56	77.96
	LoRA Rank 1	0.13745	82.45	82.67
	LoRA Rank 2	0.71656	84.36	84.89
	Propulsion(All)	<u>0.00605</u>	**89.31**	**90.61**
	Propulsion(Attn)	**0.00580**	<u>88.59</u>	<u>88.86</u>

Table 23: Entailment Classification Results for the Phi-2 Model. The best results are highlighted in **bold**, and the second-best result is <u>underlined</u> for clarity except full fine-tuning.

L Variable Description:

Variable	**Description**
$\mathbb{M}(.)$	Pre-trained language model with frozen parameters
N	Number of layers in the model
$L_i(x)$	Output of the i-th layer given input x
x	Input representation
s	Sequence length of tokens
d	Dimension of each token
V	Output of layer L_i
$\mathscr{Z}$	Trainable Propulsion matrix
$\mathbf{z_i}$	Element-wise scalar transformation vector
$\odot$	Element-wise multiplication operation
$\mathbf{v_j}'$	Transformed output after Propulsion
k	Propulsion degree for nonlinear transformation
V'	New output after Propulsion and Propulsion
$\mathscr{L}$	Cross-entropy loss function
T	Total number of data samples
$\mathbf{y}$	Ground truth labels
$\hat{\mathbf{y}}$	Predicted labels

Table 24: Table of Variables and Descriptions