CONTEXT-AWARE PROMPT TUNING: ADVANCING IN-CONTEXT LEARNING WITH ADVERSARIAL METHODS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) can perform few-shot learning using either optimization-based approaches or In-Context Learning (ICL). Optimization-based methods often suffer from overfitting, as they require updating a large number of parameters with limited data. In contrast, ICL avoids overfitting but typically underperforms compared to optimization-based methods and is highly sensitive to the selection, order, and format of demonstration examples. To overcome these challenges, we introduce Context-aware Prompt Tuning (CPT), a method inspired by ICL, Prompt Tuning (PT), and adversarial attacks. CPT builds on the ICL strategy of concatenating examples before the input, extending it by incorporating PT-like learning to refine the context embedding through iterative optimization, extracting deeper insights from the training examples. Our approach carefully modifies specific context tokens, considering the unique structure of the examples within the context. In addition to updating the context with PT-like optimization, CPT draws inspiration from adversarial attacks, adjusting the input based on the labels present in the context while preserving the inherent value of the user-provided data. To ensure robustness and stability during optimization, we employ a projected gradient descent algorithm, constraining token embeddings to remain close to their original values and safeguarding the quality of the context. Our method has demonstrated superior accuracy across multiple classification tasks using various LLM models, outperforming existing baselines and effectively addressing the overfitting challenge in few-shot learning.

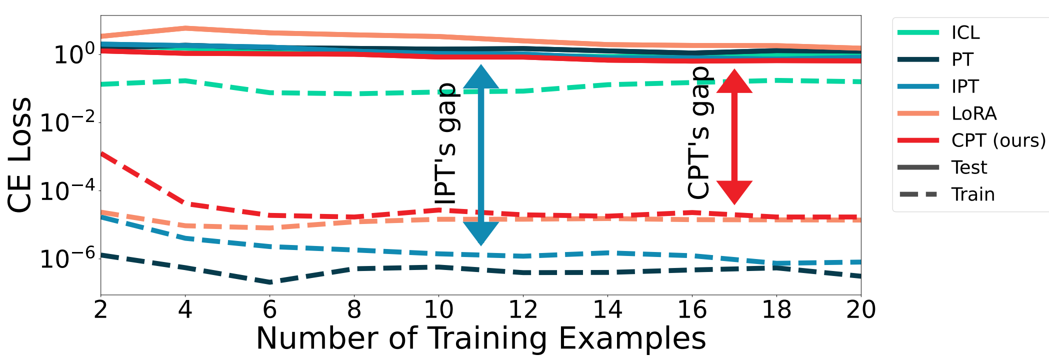

Figure 1: **Overfitting Comparison: CPT vs. Baselines** Visualizing the train-test loss gap across various methods and training set sizes using the GPT-j model on the DBpedia dataset. For each model, there are two loss graphs: one for train loss (dotted line) and one for test loss (solid line). CPT performs better in mitigating overfitting compared to optimization-based methods. Despite a relatively higher training loss, CPT achieves the lowest test loss.

[†]Department of Computer Science Technion - Israel Institute of Technology
[‡]School of Electrical and Computer Engineering - Ben-Gurion University of the Negev

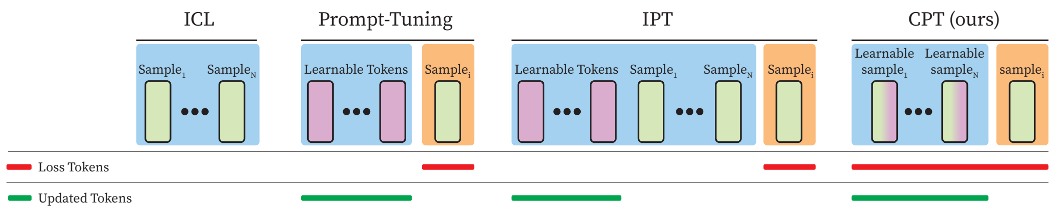

Figure 2: **Comparison of Baseline Algorithms Training and Token Utilization.** We highlight the key differences between CPT and the baselines, focusing on ICL, PT, and IPT. For each method, we emphasize two types of tokens: those used as a prefix to the input (blue background) and those used for loss (orange background). In addition, we split the prefix tokens into two groups: those updated via the training process, called *Learnable tokens* (pink), and those remaining fixed during the training process, called *Sample tokens* (green). CPT features *Sample tokens* in dual colors, reflecting their progression from *Sample tokens* to *Learnable tokens* as they are optimized.

1 Introduction

Fine-tuning Large Language Models (LLMs) is a widely used technique that adapts models to specific tasks by modifying all their parameters. Despite its effectiveness, this approach requires handling billions of parameters, which can be prohibitively expensive and inefficient, especially in terms of computational resources and storage, making it challenging to scale effectively.

To address the limitations of fine-tuning, several parameter-efficient methods have been introduced. Low-Rank Adaptation (LoRA) (Hu et al., 2021) reduces the number of trainable parameters by learning a low-rank decomposition. However, it still requires a portion of the model's weights, which remains burdensome, particularly since state-of-the-art models like Llama3 (AI@Meta, 2024) and GPT-4 (OpenAI, 2024) typically range from 7 billion to 1.7 trillion parameters. Another approach, Prompt Tuning (PT) (Lester et al., 2021), offers a more efficient solution by updating a small set of learnable token embeddings, which are concatenated before the input, while leaving the LLM's weights completely untouched. Alternatively, In-Context Learning (ICL) (Brown et al., 2020) adjusts the model to new tasks without any parameter updates, relying on the straightforward concatenation of training examples with the input context. Despite its computational efficiency, recent studies Zhang et al. (2022); Sun et al. (2023); Perez et al. (2021) indicate that ICL falls short compared to supervised fine-tuning methods. To leverage the strengths of both PT and ICL, Instruction Prompt Tuning (IPT) (Singhal et al., 2022) was introduced. This approach involves concatenating both learnable tokens and context to the input, training only the learnable tokens while keeping the context and model weights frozen.

Despite the recent advancements in parameter-efficient methods, determining the optimal method for few-shot learning remains highly unsettled. On one hand, optimization-based methods such as fine-tuning, LoRA, PT, and IPT are prone to overfitting, especially in few-shot settings where the number of trainable parameters is large – a condition known to exacerbate overfitting, as demonstrated in fig. 1. Meanwhile, In-Context Learning (ICL) mitigates overfitting by avoiding model parameter updates; however, it does not match the performance of other methods. Consequently, the most effective method for various different scenarios remains uncertain (Sun et al., 2023).

Context-aware Prompt Tuning (CPT), fuses concepts from In-Context Learning (ICL), Prompt Tuning (PT), and adversarial attacks (Blau et al., 2022; 2023; Carlini & Wagner, 2017; Athalye et al., 2018; Biggio et al., 2013; Goodfellow et al., 2014; Kurakin et al., 2016; Nguyen et al., 2015; Madry et al., 2017; Rebuffi et al., 2021; Gowal et al., 2020) into a cohesive approach, with the main differences from the baselines illustrated in fig. 2. CPT follows the ICL technique of concatenating training examples prior to the input. Similarly to PT, CPT updates only the context token embeddings through iterative optimization, leveraging again the training examples present in the context. However, CPT carefully refines the context tokens while accounting for the context's unique structure, keeping the label tokens intact, preserving their role as the ground truth. To effectively reduce overfitting and enhance performance, CPT adopts two strategies inspired by adversarial attacks: in-

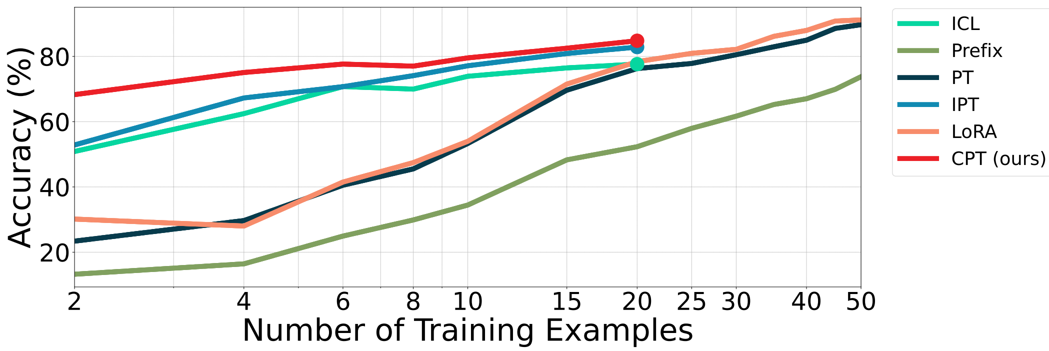

Figure 3: **Few-Shot Methods Comparison.** We compare CPT using the GPT-j model and the DBpedia dataset to baselines in few-shot settings, showing that it particularly excels when dealing with a limited number of examples. Additionally, we show that context-based methods hit memory constraints (marked with a dot) as the number of training examples rises beyond a certain level.

corporating context labels into the loss function and applying projection after updating the token embeddings. By including context labels in the loss, CPT refines input adjustments, guiding the model to optimize the entire context rather than focusing solely on the training label. To further mitigate overfitting, projected gradient descent is applied after each optimization step. This method ensures that token embedding updates remain within a controlled range, preserving proximity to their original values, under the assumption that user-provided examples are valuable. Additionally, CPT employs a loss weighting approach leverages recency bias – a phenomenon highlighted by Zhao et al. (2021), where the model tends to prioritize examples located nearer the end of the context. We recommend leveraging this property by applying an exponentially decaying weight to examples as they approach the beginning of the context, thereby increasing the emphasis on more recent examples in the optimization process.

We validate our method through a comprehensive evaluation of several classification tasks and include extensive ablations. We demonstrate that CPT outperforms other baselines across nearly every scenario, as shown in fig. 3. We use diverse templates and seeds, which is crucial due to ICL's sensitivity to training examples and format selection, as highlighted by Sun et al. (2023); Zhao et al. (2021).

To summarize, our key contributions are as follows:

- We propose a novel few-shot method called Context-aware Prompt Tuning that enhances ICL with PT and adversarial methodologies. Our method carefully optimizes the context tokens while accounting for their unique structure.
- Our method incorporates ground truth labels from the context into the loss term, optimizes with projected gradient descent, and applies recency-bias-inspired loss weighting.
- We achieve state-of-the-art results on several classification datasets and provide extensive ablation studies for each design choice of our method.

2 Related Work

Fine-Tuning Fine-tuning is a popular and effective method for adjusting LLMs to specific tasks. Standard fine-tuning (Radford et al., 2019; Brown et al., 2020; Howard & Ruder, 2018; Liu et al., 2019; Lan et al., 2019; Raffel et al., 2020; Sun et al., 2019) retrains the model with new data. However, a key disadvantage is the large number of parameters that must be stored.

Efficient Fine-Tuning To alleviate the computational burden of fine-tuning, Adapter-BERT (Houlsby et al., 2019) proposes training only the adapter layers inserted into the model, while BitFit (Zaken et al., 2021) focuses on fine-tuning just the bias terms. Delta Tuning (Ding et al., 2022) explores parameter-efficient methods that adjust only a small portion of a model's parameters. Low-

Rank Adaptation methods (LoRA) (Hu et al., 2021) introduces a novel low-rank adaptation technique, where additional low-rank matrices are added to the weights during training. This allows the model to fine-tune only these matrices, reducing the number of trainable parameters significantly. VERA (Kopiczko et al., 2023) builds on LoRA by incorporating adaptive learning rates. Compacter Karimi Mahabadi et al. (2021) leverages hypercomplex layers, and LoRA-Pro (Wang & Liang, 2024) further refines optimization. Despite these advancements, large models like GPT-3, which contain $175B$ parameters, require updating millions of parameters, such as 17.5M for LoRA.

Prompt Tuning (PT) Unlike fine-tuning methods, PT reduces the number of trainable parameters by introducing learnable tokens optimized while keeping the model's weights frozen. Lester et al. (2021) propose appending continuous prompts to the input and optimizing them, while P-tuning (Liu et al., 2023) and Prefix Tuning (Li & Liang, 2021) extend this concept by incorporating learnable tokens at intermediate layers. More recently, Wang et al. (2023) introduced the idea of training a single prompt to be shared across multiple tasks. Although these methods significantly reduce the number of trainable parameters, they face challenges in few-shot learning Gu et al. (2021) and provide limited interpretability for the learned continuous tokens (Ghosal et al., 2024; Khashabi et al., 2021; Deng et al., 2022).

In-Context Learning (ICL) In contrast to earlier methods, ICL (Brown et al., 2020) avoids optimization entirely. Instead, it concatenates task-specific examples before the input, allowing the model to learn a new task purely through observation, leveraging its pre-trained knowledge. Despite its advantages, ICL has limitations, often underperforming compared to optimization-based methods (Liu et al., 2022; Peng et al., 2023; Sun et al., 2023).

Instruction Prompt Tuning (IPT) IPT (Singhal et al., 2022) combines key elements of PT and ICL, utilizing learnable tokens that are optimized during training alongside static context tokens, similar to ICL. The concept of using both soft and hard prompts was previously introduced by PPT (Gu et al., 2021) and PTR (Han et al., 2022). Yet, IPT has struggled to consistently surpass PT in performance (Sun et al., 2023). While our method shares similarities with IPT, we focus on optimizing context tokens without introducing additional learnable tokens, and we are also leveraging context labels in the process. Another key difference lies in the optimization process, where our loss includes a regularization term, and we employ projected gradient descent to ensure the output stays close to the user-supplied reliable input.

3 Our Method

3.1 Input Preparation

Our method takes as input a few-shot classification dataset containing N examples. Each example consists of a pairing of x (an instruction) and y (a label). We embed (x, y) using input, output, and separation templates, converting them into readable text that LLMs better understand, as done in ICL Brown et al. (2020). The input and output templates, denoted T_i and T_o, along with separators S_{intra} and S_{inter}, are provided in appendix E. To embed a single example (x, y) using the template, we concatenate the input x embedded in T_i with S_{intra}, followed by the output y embedded in T_o, and finally S_{inter}, resulting in $X_{\text{Emb}_i} = [T_i(x_i), S_{\text{intra}}, T_o(y_i), S_{\text{inter}}]$. To generate the complete context, we concatenate all X_{Emb_i}, forming $X_{\text{Context}} = [X_{\text{Emb}_i}]_{i=1}^{N}$. To construct a complete training example, we randomly select an embedded example from the training set X_{Emb_i}, and concatenate it after the context, resulting $X_{\text{Train}_i} = [X_{\text{Context}}, X_{\text{Emb}_i}]$, which is then fed into the LLM. This process is also visualized in fig. 4. We supply additional concrete examples in appendix G.

Above, we described how we construct a training example X_{Train_i}, as a text sequence. However, before feeding it into the model, we must process the text through a tokenizer, which splits the text into tokens and returns an embedding vector for each token. Each example contains six types of tokens: input, input template, intra-separator, output, output template, and inter-separator. For simplicity, we ignore the separators and the fact that each part usually contains multiple tokens. For each training example i and its sub-example k, we focus on four token types: $t_{\text{I}_i}^{(k)}, t_{\text{IT}_i}^{(k)}, t_{\text{O}_i}^{(k)}, t_{\text{OT}_i}^{(k)}$, which represent the input, input template, output, and output template, respectively. Each training example i consists of $N+1$ sub-examples, N sub-examples in the context and one training sub-example at the end.

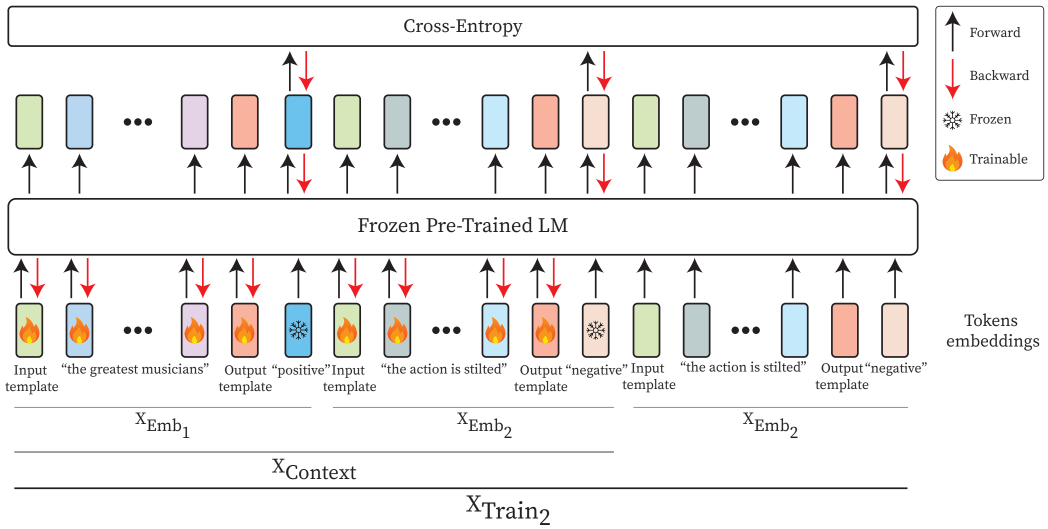

Figure 4: **Overview of CPT Training Process.** We begin by arranging the data. We concatenate all of the training examples that were embedded into the input-output templates $[X_{\text{Emb}_i}]_{i=1}^{N}$, creating X_{Context}. To this, we append a randomly selected training example, in this case X_{Emb_2}, to form the complete training example X_{Train_2}. For the training process, the input is passed through the frozen LLM, and the loss is calculated using all labels present in X_{Train_2}, covering both the context and training labels. The context is updated, but its labels remain unchanged.

3.2 OPTIMIZATION

In this section, we discuss the optimization process of our method, which is inspired by Adversarial Attacks (AT) Madry et al. (2017). Each AT step can be divided into two parts: optimization and restriction. In the first step, an attacker modifies an image to cause incorrect classification, where in the second step, the attack limits its changes to avoid detection. Similarly, our method is split into two parts: optimization, including loss design, as discussed in section 3.2.1, and controlling the token updates, as detailed in section 3.2.2.

3.2.1 LOSS DESIGN

The optimization process modifies the input embedding to help the classification. To achieve this, we introduce a new loss for each training example X_{Train_i}. This loss incorporates all the context sub-example labels X_{Context}, or more formally, we use $t_{\text{O}_i}^{(k)}$ for all $k \in [1, N]$. We use these tokens and the model's predictions for those tokens, $\hat{t}_{\text{O}_i}^{(k)}$, as shown in eq. (1).

$$L_{\text{Context}_i} = \sum_{k=1}^{N} \omega_k \cdot \text{CrossEntropy}(\hat{t}_{\text{O}_i}^{(k)}, t_{\text{O}_i}^{(k)}) \tag{1}$$

In addition to L_{Context_i}, we also apply the standard loss on the training sub-example in eq. (2).

$$L_{\text{Train}_i} = \text{CrossEntropy}(\hat{t}_{\text{O}_i}^{(N+1)}, t_{\text{O}_i}^{(N+1)}) \tag{2}$$

Lastly, we sum both losses to create the final loss $L_i = L_{\text{Context}_i} + L_{\text{Train}_i}$, where L_{Context_i} can be thought of as a regularization for the standard loss L_{Train_i}.

As explained in section 3.1, each training example X_{Train_i} contains $N + 1$ sub-labels, from N sub-examples in the context and one training sub-example. However, not all sub-examples should be weighted equally. For instance, the last sub-example is more important as it is located in the location of the test examples. Additionally, sub-examples closer to the end of the context carry more importance (Zhao et al., 2021). Thus, we apply exponential loss weight decay starting from the end of the context and decaying towards the beginning, while keeping L_{Train_i} unchanged. Formally, each sub-example k is multiplied by γ^j, where $j = N + 1 - k$. For example, the last sub-example is multiplied by γ^1, and the second-to-last by γ^2, and so on. The decay is shown in eq. (1) as ω_k.

3.2.2 Controlled Token Embedding Optimization

As mentioned in section 3.2.1, we use all the labels in each X_{Train_i} to optimize the tokens within the context. However, some tokens in the context represent labels, and we do not update these label tokens, as they carry valuable ground truth information. Instead, we update the other tokens in the context, carefully managing these updates to ensure controlled and precise modifications.

The controlled modification is designed with two key objectives. First, we trust the user to provide meaningful examples representing the task, so the context should stay close to the user's intent, minimizing significant changes. Second, few-shot optimization can lead to overfitting without proper regularization. Controlled modification addresses both issues: it acts as a regularization mechanism while preventing overfitting. For instance, as changes become smaller, our method converges to ICL, which is robust against overfitting. We achieve this by using projected gradient descent, which limits each token's embedding change to an ℓ_2 norm of size ϵ after each optimization step. Further explanations are provided in appendix H.

4 Experimental Setup

In this section, we provide details regarding the datasets, models, baselines, and evaluation used in our experiments. Implementation details are provided in appendix F.

4.1 Datasets

In this work, we focus on a classification task and select a variety of datasets to ensure robust conclusions across different task types. We include SST-2 (Socher et al., 2013) for sentiment analysis, AG News (Zhang et al., 2015b) for news classification, DBpedia (Zhang et al., 2015a) for ontology classification, and TREC (Li & Roth, 2002) for question classification. These datasets represent a diverse range of natural language classification tasks, include different number of classification classes, allowing us to evaluate our method comprehensively. More details are provided in appendix D.

4.2 Models

We use models of varying sizes and quality to ensure robust evaluation and conclusions. For the relatively small model, we use BLOOM1.7B (Scao et al., 2022), while for larger models, we opt for GPT-j6B(Wang & Komatsuzaki, 2021) and Llama3 8B(AI@Meta, 2024). The GPT-j model is noted for its robust performance, while Llama3 is currently among the leading models in the field.

4.3 Baselines

We compare our method to several groups of few-shot learning techniques. In the first group, we include LoRA (Hu et al., 2021), one of the leading efficient fine-tuning methods. Additionally, we compare against several prompt-tuning approaches, including Prompt Tuning (PT) (Lester et al., 2021), Prefix Tuning (Li & Liang, 2021), and Instruction Prompt Tuning (IPT) (Singhal et al., 2022). Finally, we compare our method to In-Context Learning (ICL) (Brown et al., 2020).

For some of the few-shot methods, we introduce an alternative version that incorporates instructions, as indicated in table 1 with a †. Instead of initializing the learnable tokens randomly, we initialize them with instructions specified in appendix C. We apply instructions to PT, IPT, and our method, reporting results for both random and instruction-based prompt initialization. An example is provided in appendix G.

Table 1: **Baseline Comparisons** Mean accuracy of various methods and our CPT, across several models and datasets. Evaluations are conducted using 2, 4, and 6 shots.

Dataset	Method	BLOOM 1.7B			GPT-j 6B			Llama3 8B		
		2	4	6	2	4	6	2	4	6
SST-2	Prefix	47.80	47.33	49.00	52.23	52.50	52.87	–	–	–
	ICL	50.53	60.83	61.87	50.57	67.47	77.47	76.43	80.63	83.10
	PT†	64.97	65.07	65.07	57.10	52.93	55.70	72.97	73.47	84.57
	PT	56.03	56.90	58.33	64.07	64.37	64.60	64.27	65.70	67.03
	IPT†	58.50	61.83	62.80	51.50	**83.20**	84.80	86.90	88.03	94.40
	IPT	48.50	58.80	61.87	48.13	82.27	87.17	57.20	87.40	90.43
	LoRA	**66.40**	66.93	66.90	**69.80**	71.53	73.17	68.73	71.27	83.97
	CPT†	59.53	**72.40**	**74.83**	52.53	82.03	**88.07**	**92.73**	95.07	96.40
	CPT	50.77	70.70	74.10	50.53	82.90	88.03	83.83	**96.30**	**96.50**
AG News	Prefix	24.87	25.35	26.02	32.32	33.33	46.08	–	–	–
	ICL	35.12	34.28	42.48	66.73	62.38	69.57	79.38	82.32	85.27
	PT†	28.67	30.73	41.17	37.85	44.85	62.92	59.60	57.02	68.02
	PT	33.57	36.98	**56.08**	56.85	56.13	75.10	69.32	67.92	69.33
	IPT†	36.95	31.90	42.93	67.02	63.00	74.85	82.93	**84.45**	85.08
	IPT	38.77	38.20	47.78	66.02	63.92	74.00	80.52	76.30	80.98
	LoRA	29.50	30.80	33.98	56.12	56.03	72.55	70.62	74.97	73.70
	CPT†	33.68	33.13	41.10	71.35	**68.73**	75.68	83.17	84.28	84.67
	CPT	**40.85**	**44.48**	50.40	**74.80**	68.62	**76.22**	**83.78**	81.92	**85.43**
DBpedia	Prefix	19.76	19.74	23.65	13.25	16.43	24.94	–	–	–
	ICL	48.20	51.40	55.17	50.87	62.46	70.76	71.66	72.44	79.93
	PT†	24.90	26.32	34.75	21.01	22.12	37.44	55.30	57.21	66.26
	PT	46.71	41.94	45.93	23.39	29.69	40.53	55.81	52.72	55.02
	IPT†	33.28	40.36	45.85	47.10	67.60	75.09	81.10	87.69	92.06
	IPT	48.09	54.60	70.57	52.86	67.27	70.73	72.92	76.11	78.44
	LoRA	43.30	41.13	41.18	30.15	28.02	41.50	54.24	59.50	63.21
	CPT†	33.80	48.13	51.18	53.20	**77.30**	**81.00**	**84.23**	**90.33**	**93.08**
	CPT	**58.85**	**65.78**	**73.55**	**68.29**	75.07	77.65	77.38	78.49	82.42
TREC	Prefix	19.10	24.49	29.92	30.76	30.04	27.87	–	–	–
	ICL	33.54	33.33	28.53	28.94	35.14	32.49	35.32	42.48	40.34
	PT†	30.91	33.70	39.31	29.02	34.66	43.89	43.42	48.81	51.73
	PT	32.18	32.26	35.69	31.16	32.79	37.86	32.77	33.98	33.83
	IPT†	27.83	36.64	42.92	31.04	43.12	43.09	51.72	62.14	65.13
	IPT	32.37	36.59	42.60	29.59	38.90	40.38	36.94	45.62	52.08
	LoRA	34.07	33.22	33.50	34.17	33.73	37.63	31.21	33.21	36.36
	CPT†	29.72	35.64	**45.38**	33.39	44.20	**45.83**	**57.26**	**67.00**	**69.29**
	CPT	**35.68**	**41.79**	45.16	**35.37**	**44.66**	42.71	45.12	57.54	60.18

4.4 Evaluation

We evaluate each model and dataset using three different numbers of training samples: 2, 4, and 6. For each configuration, the reported results are averaged accuracy over 30 experiments, consisting of 10 randomly sampled templates and 3 different random seeds, with the templates described in appendix E. By utilizing randomized seeds, we ensure variation in the selection of training examples. This extensive setup is crucial for achieving a comprehensive and robust evaluation, especially given that these methods are known to be highly sensitive to the selection of training examples and templates (Voronov et al., 2024; Zhao et al., 2021). Further evaluation details can be found in appendix B.

5 RESULTS

5.1 MAIN RESULTS

In table 1, we demonstrate that CPT convincingly performs better than the baselines in most cases, with particularly pronounced gains in harder tasks. Furthermore, CPT 's performance becomes more efficient and effective as the models grow stronger, such as with Llama3.

Performance on Challenging Tasks CPT demonstrates improvements across various datasets, with more pronounced gains in tasks we define as harder based on two factors: the number of shots and the number of classes. As illustrated in table 1, task difficulty increases with fewer shots and more classes. For example, on the DBpedia dataset, which has 14 classes, decreasing the shots from 6 to 4 widens the performance gap between CPT and the baselines from (3, 6, 1) to (11, 10, 3) across the models: BLOOM, GPT-j, and Llama3.

Decisive Advantage with Powerful Models The strength of the model plays a significant role in performance. As the model becomes better, CPT's advantage becomes more pronounced across all datasets and shot settings. For instance, Llama3 consistently outperforms other baselines across all datasets, except in one case where results are comparable. With GPT-j, a slightly older model, the results are lower in two instances, with one comparable outcome, both on SST-2 , the easier task as previously discussed. When comparing with BLOOM , the weakest model in our comparison, we observe lower performance on two occasions, specifically on the two easier datasets.

5.2 STANDARD DEVIATION

Standard deviation (std) plays a crucial role in few-shot learning due to the sensitivity of these methods to both the training examples and the chosen template (Zhao et al., 2021; Voronov et al., 2024). In fig. 5, we present accuracy along with two types of std bars: black bars represent the mean std across different templates, while blue bars represent the mean std across different seeds. We demonstrate that CPT significantly improves accuracy across various models and datasets in a statistically significant manner. More information is presented in appendix A.

Our method's standard deviation performs equivalently to other methods in most cases, while in certain cases, such as with DBpedia, CPT exhibits both higher accuracy and lower std, reinforcing its robustness in complex tasks. However, the sensitivity of our method does not follow a clear pattern across random seeds or templates. For instance, while randomness in templates and training examples has an equal influence on std in DBpedia and TREC , SST-2 shows a higher std for template randomness, and AG News is more sensitive to variations in training examples.

5.3 ABLATIONS

The most important design choices that positively impacted CPT's performance are the loss design and the projections. These improvements are evident across 2, 4, and 6-shot settings, as shown in

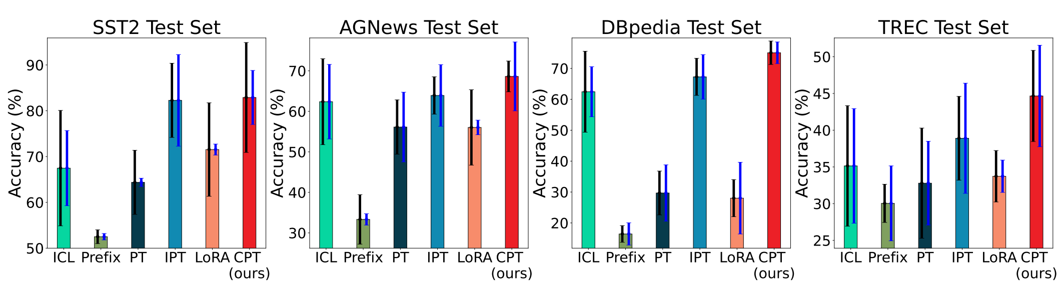

Figure 5: **Accuracy and Standard Deviation** Comparison of accuracy and standard deviation between CPT and baselines, evaluated with 4-shot on GPT-j model. The black bars represent the mean std across different templates, while the blue bars represent the mean std across different seeds.

Table 2: **Ablation Study** We present the mean accuracy for various ablations using the GPT-j model and the DBpedia dataset, including loss tokens (train example, random, or all context), loss weighting (decay and mean), projection type (token-wise or all-tokens), epsilon values for input and format, updated tokens (input, format, masks), and masking of the training example.

Loss Tokens	Loss Weighting	Projection Type	Input ϵ	Format ϵ	Updated Tokens	Mask Training Example	Number of Training Examples 2	4	6
Train Example							58.09	61.54	66.69
Train Example & 1 Random	Decay 0.95	Token-Wise	0.1	0.1	Input & Format	✗	69.48	72.08	76.80
Train Example & All Context							69.54	73.03	76.58
Train Example & All Context	Mean	Token-Wise	0.1	0.1	Input & Format	✗	69.62	72.91	76.49
	Equal 1						69.07	72.82	76.23
	Equal 10						69.35	71.01	75.11
	Decay 0.99						69.59	72.97	76.43
	Decay 0.95						69.54	73.03	76.58
	Decay 0.5						69.60	72.39	76.44
Train Example & All Context	Decay 0.95	All-Tokens	0.001	-	Input & Format	✗	51.52	63.41	71.50
			0.01	-			56.37	68.12	73.66
			0.1	-			69.51	72.64	76.06
			1.0	-			63.11	64.78	71.94
Train Example & All Context	Decay 0.95	Token-Wise	0.01	0.1	Input & Format	✗	65.61	70.12	75.63
			0.1	0.1			69.54	73.03	76.58
			1.0	0.1			65.29	66.30	73.63
			0.1	0.01			69.53	73.55	76.55
			0.1	1.0			68.27	71.91	68.27
Train Example & All Context	Decay 0.95	Token-Wise	0.1	0.1	Input	✗	69.47	74.13	76.63
					Masks		63.74	69.21	74.91
					Input & Format		69.54	73.03	76.58
Train Example & All Context	Decay 0.95	Token-Wise	0.1	0.1	Input & Format	✓	67.55	64.26	68.58

table 2. Different options for the loss design are specified under "Loss Tokens", with three configurations: using only the training label, using the training label plus one random context label, and using the training label plus all context labels. The latter significantly outperforms the training-only configuration. The ablation over the projection is specified under "Input ϵ" and "Format ϵ" demonstrating that both too small a change (which converges to ICL) and too large a norm are suboptimal. Lastly, we examined the effect of "Loss Weighting". We propose three options: Mean, which applies uniform weighting across all labels; Equal, which assigns equal weight to both the training label loss and the context label losses; and the option we use, Decay, which reduces the influence of context labels further from the training example. On this dataset, Decay works slightly better, and in most cases, the improvement is more significant.

In addition to these core design choices, we explored several alternative configurations that ultimately did not enhance performance. Under "Loss Weighting", we experimented with the "Equal" option, which assigns equal loss weight to both the training example loss and the entire context loss, where the training loss can be multiplied by the noted value (*e.g.*, 1, 10). We also tested the projection type "All-Tokens" which applies the projection to the entire context collectively rather than token-by-token. Under "Updated Tokens" we attempted to modify only specific parts of the context. Additionally, under "Mask Training" we masked the training example from the context to prevent the model from simply copying the answer. However, none of these configurations led to performance improvements. Additional ablation experiments are presented in appendix I.

6 Discussions

CPT demonstrates significant advancements in few-shot learning by integrating ICL with PT and adversarial strategies, refining context embeddings. Unlike traditional fine-tuning and other parameter-efficient approaches, CPT optimizes only the context tokens, making it particularly effective in few-shot settings, where overfitting is a concern. CPT achieves improved generalization across a wide variety of tasks, demonstrates a significant advancement, and offers meaningful insights for future few-shot learning methods.

Limitation & Future Work The computational cost associated with the iterative optimization of context embeddings is significant compared to ICL. Additionally, similar to ICL and IPT, CPT is limited in the number of examples it can handle, as memory consumption scales with context length. In contrast, traditional methods are better suited for larger datasets. Future work could explore more efficient optimization strategies to reduce computational overhead and improve scalability.

REFERENCES

AI@Meta. Llama 3 model card. 2024. URL https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pp. 284–293. PMLR, 2018.

Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pp. 387–402. Springer, 2013.

Tsachi Blau, Roy Ganz, Bahjat Kawar, Alex Bronstein, and Michael Elad. Threat model-agnostic adversarial defense using diffusion models. *arXiv preprint arXiv:2207.08089*, 2022.

Tsachi Blau, Roy Ganz, Chaim Baskin, Michael Elad, and Alex Bronstein. Classifier robustness enhancement via test-time transformation. *arXiv preprint arXiv:2303.15409*, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pp. 3–14, 2017.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric P Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. *arXiv preprint arXiv:2205.12548*, 2022.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models. *arXiv preprint arXiv:2203.06904*, 2022.

Soumya Suvra Ghosal, Samyadeep Basu, Soheil Feizi, and Dinesh Manocha. Intcoop: Interpretability-aware vision-language prompt tuning. *arXiv preprint arXiv:2406.13683*, 2024.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Sven Gowal, Chongli Qin, Jonathan Uesato, Timothy Mann, and Pushmeet Kohli. Uncovering the limits of adversarial training against norm-bounded adversarial examples. *arXiv preprint arXiv:2010.03593*, 2020.

Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. Ppt: Pre-trained prompt tuning for few-shot learning. *arXiv preprint arXiv:2109.04332*, 2021.

Xu Han, Weilin Zhao, Ning Ding, Zhiyuan Liu, and Maosong Sun. Ptr: Prompt tuning with rules for text classification. *AI Open*, 3:182–192, 2022.

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

Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

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

Rabeeh Karimi Mahabadi, James Henderson, and Sebastian Ruder. Compacter: Efficient low-rank hypercomplex adapter layers. *Advances in Neural Information Processing Systems*, 34:1022–1035, 2021.

Daniel Khashabi, Shane Lyu, Sewon Min, Lianhui Qin, Kyle Richardson, Sean Welleck, Hannaneh Hajishirzi, Tushar Khot, Ashish Sabharwal, Sameer Singh, et al. Prompt waywardness: The curious case of discretized interpretation of continuous prompts. *arXiv preprint arXiv:2112.08348*, 2021.

Dawid J. Kopiczko, Tijmen Blankevoort, and Yuki M. Asano. Vera: Vector-based random matrix adaptation, 2023.

Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

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

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

Xin Li and Dan Roth. Learning question classifiers. In *COLING 2002: The 19th International Conference on Computational Linguistics*, 2002. URL https://www.aclweb.org/anthology/C02-1150.

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

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

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

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

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

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 427–436, 2015.

OpenAI. Chatgpt (september 19 version). https://chat.openai.com, 2024. Large language model.

Hao Peng, Xiaozhi Wang, Jianhui Chen, Weikai Li, Yunjia Qi, Zimu Wang, Zhili Wu, Kaisheng Zeng, Bin Xu, Lei Hou, et al. When does in-context learning fall short and why? a study on specification-heavy tasks. *arXiv preprint arXiv:2311.08993*, 2023.

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. True few-shot learning with language models. *Advances in neural information processing systems*, 34:11054–11070, 2021.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Nir Ratner, Yoav Levine, Yonatan Belinkov, Ori Ram, Inbal Magar, Omri Abend, Ehud Karpas, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. Parallel context windows for large language models. *arXiv preprint arXiv:2212.10947*, 2022.

Sylvestre-Alvise Rebuffi, Sven Gowal, Dan A Calian, Florian Stimberg, Olivia Wiles, and Timothy Mann. Fixing data augmentation to improve adversarial robustness. *arXiv preprint arXiv:2103.01946*, 2021.

Teven Le Scao, Angela Fan, Christopher Akiki, and et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022.

Karan Singhal, Shekoofeh Azizi, Tao Tu, S Sara Mahdavi, Jason Wei, Hyung Won Chung, Nathan Scales, Ajay Tanwani, Heather Cole-Lewis, Stephen Pfohl, et al. Large language models encode clinical knowledge. *arXiv preprint arXiv:2212.13138*, 2022.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D13-1170.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *Chinese computational linguistics: 18th China national conference, CCL 2019, Kunming, China, October 18–20, 2019, proceedings 18*, pp. 194–206. Springer, 2019.

Simeng Sun, Yang Liu, Dan Iter, Chenguang Zhu, and Mohit Iyyer. How does in-context learning help prompt tuning? *arXiv preprint arXiv:2302.11521*, 2023.

Anton Voronov, Lena Wolf, and Max Ryabinin. Mind your format: Towards consistent evaluation of in-context learning improvements. *arXiv preprint arXiv:2401.06766*, 2024.

Ben Wang and Aran Komatsuzaki. Gpt-j-6b: A 6 billion parameter autoregressive language model. https://github.com/kingoflolz/mesh-transformer-jax, 2021. Accessed: 2024-05-26.

Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, Huan Sun, and Yoon Kim. Multitask prompt tuning enables parameter-efficient transfer learning. *arXiv preprint arXiv:2303.02861*, 2023.

Zhengbo Wang and Jian Liang. Lora-pro: Are low-rank adapters properly optimized? *arXiv preprint arXiv:2407.18242*, 2024.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing, 10 2020. URL https://github.com/huggingface/transformers.

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

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015a. URL https://proceedings.neurips.cc/paper_files/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *NIPS*, 2015b.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. Calibrate before use: Improving few-shot performance of language models. In *International conference on machine learning*, pp. 12697–12706. PMLR, 2021.