PromptIntern: Saving Inference Costs by Internalizing Recurrent Prompt during Large Language Model Fine-tuning

Anonymous ACL submission

Abstract

Large language models (LLMs) have played a fundamental role in various natural language processing tasks with powerful prompt techniques. However, in real-world applications, there are often similar prompt components for repeated queries, which causes significant computational burdens during inference. Existing prompt compression and direct fine-tuning methods aim to tackle these challenges, yet they frequently struggle to strike an optimal balance between cost-efficiency and performance effectiveness, especially in complex tasks such as NL2Code. In this paper, we propose a novel method namely *PromptIntern* to internalize the prompt knowledge into model parameters via progressive fine-tuning. Our method enables LLMs to emulate the human learning process for a new task, where detailed templates and examples in a prompt are gradually internalized and phased out progressively as the model grows accustomed to the task. Extensive experiments demonstrate that our method can reduce the inference tokens by 67-90%, saves 39-90% cost, and speedups inference by 1.1-5.1x.

1 Introduction

Large language models (LLMs) have become pivotal in numerous natural language processing (NLP) applications, such as natural language generation (Dong et al., 2019), reasoning (Zhu et al., 2023; Sui et al., 2023), and code generation (Luo et al., 2023; Li et al., 2023a; Rozière et al., 2024). In practical deployments, crafting suitable prompts is of great importance as it can substantially improve the prediction performance. Advanced prompt engineering techniques, such as chain-of-thought prompting (Wei et al., 2022), self-consistency (Wang et al., 2022), and retrieval-augmented generation (Lewis et al., 2020), have significantly advanced the capabilities of LLMs. However, these techniques often involve in much longer prompts, which substantially increases the

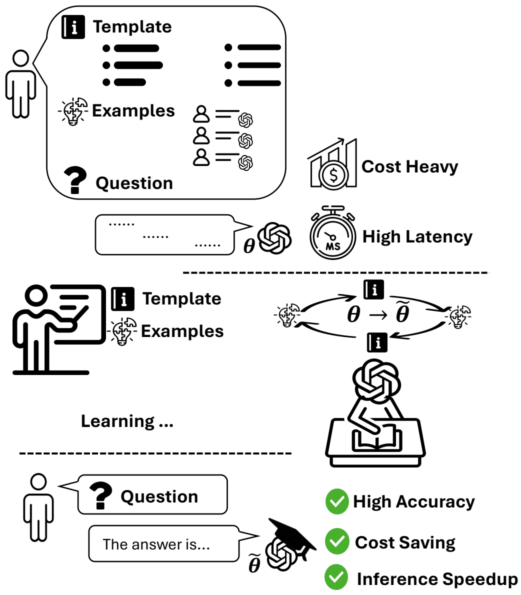

Figure 1: An illustration of PromptIntern.

computational cost during inference. Such an increase in inference cost can preclude the application of LLMs in many cost-sensitive scenarios where computational resources are constrained.

To mitigate the substantial inference cost incurred by advanced prompt engineering techniques, numerous methods (Jiang et al., 2023a,b; Pan et al., 2024) have been proposed to compress prompts while minimizing performance degradation. However, existing compression methods have primarily focused on relatively straightforward tasks, such as text summarization (Zhang et al., 2019), where significant neural language redundancy exists. For more challenging tasks, especially those involving knowledge contained within examples, the corresponding tokens are inherently more difficult to compress. Naively applying compression techniques to these knowledge-intensive prompts can lead to significant performance drops, as the relevant information may be inadvertently removed or distorted during the compression process. On the other hand, fine-tuning provides a straightforward way to enhance the model's performance for a specific task. However, direct fine-tuning without the guidance of prompt instruction or examples often suffers from significant performance degradation, posing a serve challenge to the training process.

In this paper, we propose a novel paradigm to internalize the prompt input and enable efficient inference. Instead of directly removing prompt tokens based on their perplexity or compressing them into smaller tokens, we aim to transfer the prompt knowledge into model parameters. Our idea is motivated by the human learning process as illustrated in Figure 1: When a human **intern** is first exposed to a new task, they typically require detailed instructions, demonstrative examples, and relevant documents to effectively understand and **intern**alize the task knowledge. However, as the intern becomes accustomed to the task, such guidance is no longer needed, as he has mastered the required knowledge. Similarly, for LLMs, when similar knowledge is repeatedly exposed to the model, the LLM should gradually learn and internalize it into its parameters. This means that when faced with new, similar tasks, the model should be able to predict accurately without the need for extensive prompting as initially.

To achieve this goal, we delineate the prompt into three distinct components: the template, examples, and query, and propose a progressive fine-tuning strategy that involves gradually compressing the prompt template and reducing the number of retrieved examples, enabling the model to incrementally absorb the prompt knowledge into its parameters. We further design tailored compression strategies for different components. We dub our approach "**PromptIntern**" as it **intern**alizes prompt knowledge into LLM models, viewing the LLM as a human intern learning tasks progressively.

PromptIntern can been conceptually connected to curriculum learning, where the model is initially presented with relatively straightforward samples accompanied by complete prompt contexts. Subsequently, the prompt instructions and examples are progressively compressed, gradually increasing the task difficulty. Through this progressive exposure and fine-tuning process, we are able to foster better model learning capabilities, resulting in improved zero-shot performance. As a result, we are able to maximize prompt compression while preserving satisfactory performance, striking an optimal balance between inference efficiency and accuracy. We conduct extensive experiments on challenging code generation tasks to demonstrate the efficiency and effectiveness of our approach. Our main contributions can be summarized as follows:

- We propose *PromptIntern*, a novel prompt compression method that aims to internalize repetitive prompt knowledge into the model's parameters, achieving extreme inference efficiency while maintaining high performance.
- We devise detailed prompt internalization strategies for different prompt components along with a tailored progressive fine-tuning pipeline.
- We conduct extensive experiments with detailed analysis on challenging NL2Code tasks. The experimental results demonstrate that our approach reduces the tokens usage by 67-90%, achieves 39-90% cost savings, and speedups inference by 1.1-5.1x.

2 Related Work

Prompt compression aims to rephrase the original prompts into condense ones. Depending on whether targeting on specific tasks, it can be categorized into task-aware and task-agnostic compression approaches. Task-aware approaches compress the context based on the downstream tasks or current query. For example, LongLLMLingua (Jiang et al., 2023b) adopts a question-aware coarse-to-fine approach based on the information entropy of the tokens and adapts the information according to the question. Soft prompt methods (Wingate et al., 2022; Liu et al., 2022; Mu et al., 2024) condense the input prompt with learnable tokens. Task-agnostic approaches typically involve using information entropy-based metrics to remove redundant information in the prompt (Li et al., 2023b; Jiang et al., 2023a). For example, LLMLingua (Jiang et al., 2023a) uses a small language model to estimate token importance. Despite their demonstrated effectiveness, producing compressed text that can generalize across different tasks remains a challenge (Pan et al., 2024). Different from existing prompt compression methods, we propose internalizing the prompt knowledge into model parameters to handle repetitive queries, thereby enabling a higher degree of inference efficiency.

Model fine-tuning aims to adapt the pretrained LLM model to specific downstream tasks by modifying the model parameters. Based on the assumption that fine-tuning adds less new information to the model pretrained on large Inernet-scale datasets, Parameter-Efficient Fine-Tuning (PEFT) methods are designed to reduce the high expense of fine-tuning large-scale models. PEFT achieves this by training only a small subset of the model's total parameters to adapt to the new task. Existing PEFT methods can be broadly categorized into three main approaches: 1) Adapter-based methods (Houlsby et al., 2019; He et al., 2021): These introduce additional trainable modules into a frozen "backbone" network. This offers flexibility but can increase the model size. 2) Prompt-based methods (Lester and Constant; Razdaibiedina et al., 2023; Nashid et al., 2023): These introduce additional trainable "soft tokens" at the beginning of the input sequence. This is simpler but might require crafting effective prompts for each task. 3) Low-rank adaptation methods (Hu et al., 2021; Dettmers et al., 2024; Liu et al., 2024): These utilize low-rank matrices to approximate the weight changes needed for fine-tuning. This is the current mainstream approach because it avoids adding any burden during inference and often exhibits strong performance.

3 Problem Formulation

In this paper, we define a input prompt as $x = (x^{tmp}, x^{egs}, x^{que})$, where each input prompt x is considered as a tuple of three components: x^{tmp} as the template such as fixed instructions, API docs, etc., x^{egs} as the examples, and x^{que} as the query. Typically, x^{tmp} and x^{egs} are relatively fixed and lengthy but essential for complex tasks. Let $f_\theta(\cdot)$ denotes the neural network function of a LLM model, typically transformer (Vaswani et al., 2017), parameterized by θ. The generated output by LLM can be represented as $f_\theta(x)$.

We then consider the following problem of prompt internalization. Given a training dataset $\mathcal{D}_{train} = \{(x_i, y_i)\}_{i=1}^n$ where n is the number of training samples, x_i is an input prompt defined above, and y_i is the corresponding groundtruth output. Our goal is to internalize the knowledge contained in templates and examples of each input prompt i.e. $\{(x_i^{tmp}, x_i^{egs})\}_{i=1}^n$ into model parameters θ during fine-tuning, enabling efficient inference while maintaining high prediction performance through $\{x_i^{que}\}_{i=1}^n$ only. Formally, the prompt internalization objective can be formulated as follows:

$$\min_{\tilde{\theta}} \sum_{i=1}^{n} \mathcal{L}\left(y_i, f_{\tilde{\theta}}(x_i^{que})\right) \tag{1}$$

where $\mathcal{L}(\cdot)$ denotes the loss function and $\tilde{\theta}$ denotes the updated weights with internalized prompt knowledge. For a new incoming prompt only containing the query, the updated LLM with $f_{\tilde{\theta}}(\cdot)$ can internally recover the output without the assistance of instruction and examples.

4 Methodology

In this section, we introduce the detailed procedures of PromptIntern. We first discuss the template compression that is designed to compress the entire fixed template part inside an input prompt. Then we describe the example absorption on how to effectively absorb demonstration examples into model parameters. Finally, we introduce a tailored training strategy for PromptIntern. The overall framework is shown on Figure 2.

4.1 Template Compression

We first introduce template compression, which is designed to compress the common template information existed across training instances. The motivation of the template compression stems from the following aspects: 1) Redundancy. The instruction is repetitive across prompts for a given task, often containing unnecessary tokens that do not contribute to the language model's understanding, posing significant memory and computational burdens when the instruction is lengthy; and 2) Noise. Excessively long prompts may incorporate extraneous elements—either irrelevant or misleading information—that serves as noise and can adversely affect the model's generation.

To mitigate the issues stated above, we propose a template compression system, which can generally be expressed as:

$$\tilde{x}^{\text{tmp}} = C(x^{\text{tmp}}, \tau^{\text{tmp}}) \tag{2}$$

where C is a specific template compressor, $\tilde{x}^{tmp}$ is the compressed template, and τ^{tmp} is the template compression rate as defined in (Jiang et al., 2023a), varying at diffent training interations. We then adopt a predetermined schedule $\mathcal{S}^{tmp}(t)$ to progressively reduce and internalize the prompt template information during the t-th training iteration. Specifically, for a total of T training iterations, we initially set τ^{tmp} to 1 at $\mathcal{S}^{tmp}(0)$ and gradually decrease the value of τ^{tmp} at $\mathcal{S}^{tmp}(t)$ to zero at end to achieve fully template internalization. Note that such compression system is also flexible, allowing it to halt at a desired non-zero compression

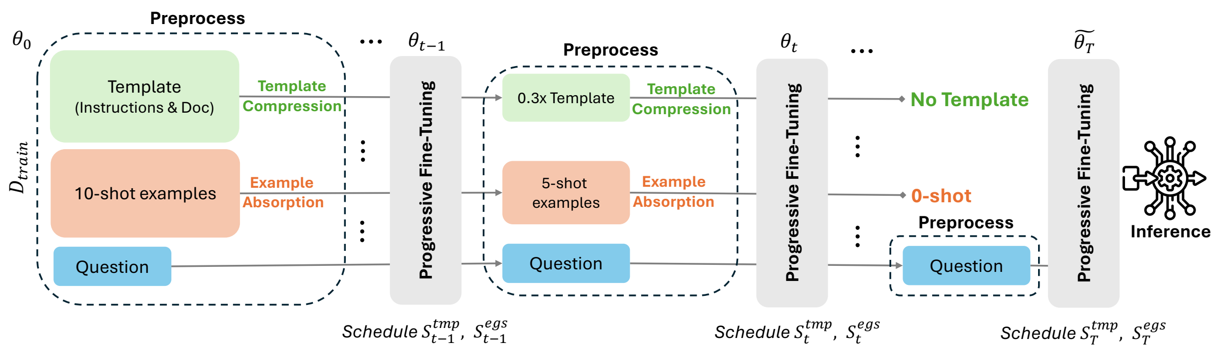

Figure 2: Overview of our PromptIntern pipeline. We adopt a progressive fine-tuning approach to gradually internalize the prompt knowledge that exists in the template and examples into the model parameters. In this way, we can perform efficient inference without compromising performance compared to regular few-shot fine-tuning.

rate. This flexibility allows to maintain a certain level of compressed template, serving as a trade-off to preserve inference accuracy in specific scenarios, as discussed in Section 5.3. In addition of the progressively decreasing template schedule, we also specify the template compressor C for better utilization. we categorize it into two types which exactly reflects the primary components of the template defined in the problem formulation: the instruction compressor and document compressor:

1) Instruction Compressor targets the static elements within prompts, specifically focusing on the instructional content. Instructions in training data often consist of repeated directives, guidelines, or predefined tasks which are common across multiple training scenarios. The primary goal of the instruction compressor is to distill these instructions down to their essential components, eliminating verbosity and redundancy without compromising the clarity or intent of the instructions.

2) Document Compressor is designed to handle the bulkier and more detailed portions of the prompts, such as API documentation or static demonstrations. These sections typically include extensive technical descriptions and examples that, while informative, often contain a significant amount of repetitive or non-essential information (Xu et al., 2023). The goal of the document compressor is to reduce the information unnecessary for understanding and applying the technical content, thereby streamlining the training process.

4.2 Example Absorption

Incorporating few-shot examples into fine-tuning not only improves information retrieval and memory recall (Hübotter et al., 2024) but also yields substantial benefits in handling a variety of tasks with minimal data input (Mosbach et al., 2023; Snell et al., 2017). However, directly adding lengthy few-shot examples to input prompts burdens the context window and increases inference latency. Motivated by this, we propose example absorption to benefit from the enhanced performance afforded by few-shot examples while prevent incurring significant additional overhead. Specifically, the example absorption mainly contains two stages: example retrieval and example removal.

1) Example Retrieval is designed to identify and select the most related few-shot examples from the training dataset and incorporate them into each training instance. The underlying rationale is to choose examples that closely align with the training instance so as to accelerate model's internalization during training. We employ a straightforward approach that utilizes a relevance scoring function $s(\cdot, \cdot)$ to assess the similarity between examples and the training instance. Specifically, we select the top k examples, varying at different training iterations, with the highest relevance scores to serve as our few-shot examples. For a training instance (x_i, y_i) with x_i being the input prompt and y_i being the corresponding groundtruth output, the selection process can be expressed as follows:

$$x_i^{egs} = \{(x_j, y_j) \mid j \neq i, s(y_i, y_j) \in \text{top } k \text{ scores}\} \tag{3}$$

Note that the scoring function is calculated based on common similarity metrics (Rubin et al., 2022; Chen et al., 2022; Dai et al., 2022). In our experiment, we use the BLEU as the scoring function.

2) Example Removal aims to progressively internalize the prompt knowledge from few-shot examples into model parameters. To achieve this, we also adopt a predetermined schedule $\mathcal{S}^{egs}(t)$

Algorithm 1 PromptIntern Pipeline

Input: A training dataset $\mathcal{D}_{train} = \{(x_i, y_i)\}_{i=1}^{n}$ with $x_i = (x_i^{tmp}, x_i^{egs}, x_i^{que})$ and corresponding labels y_i, A language model f with initial parameters θ, learning rate η, training iterations T, template compression schedule $\mathcal{S}^{tmp}$, example absorption schedule $\mathcal{S}^{egs}$

Output: The inference output $f_{\theta_T}(x^{que})$

```
1:  Preprocess
2:  for i = 1, 2, ..., n do
3:      Obtain each τ^tmp from S^tmp
4:      Obtain each k from S^egs
5:      Compress x_i^tmp w/ each τ^tmp via Eq. (2)
6:      Retrieve k examples x_i^egs via Eq. (3)
7:  end for
8:  Progressive Finetuning
9:  for t = 0, 1, ..., T − 1 do
10:     Adjust prompts with S^tmp(t) and S^egs(t)
11:     Update model parameters θ_t via Eq. (4)
12: end for
13: Inference
14: Perform inference with f_{θ_T}(x^que)
```

to gradually decrease the number of demonstration examples in each prompt instance during the t-th iteration. Specifically, for a total of T training iterations, we initially set k examples at $\mathcal{S}^{egs}(0)$ and then gradually decrease the value of k at each $\mathcal{S}^{egs}(t)$ to zero at end in order to achieve fully example internalization.

4.3 PromptIntern Pipeline

In this subsection, we describe the detailed pipeline of PromptIntern. As demonstrated in Algorithm 1, PromptIntern consists of three stages: preprocess (line 1-5), progressive fine-tuning (line 6-10), and inference (line 11-12).

1) Preprocess. We first preprocess the input prompts to prepare them for the progressive training stage. Specifically, we process the prompt template to different compression rates based on the schedule $\mathcal{S}^{tmp}(t)$ and retrieve examples for each training instance based on the schedule $\mathcal{S}^{egs}(t)$. For better illustration, we provide an example of a pre-processed prompt with respect to schedule in Figure 3.

2) Progressive Fine-tuning. We then fine-tune the model parameters for internalizing. Given the training iteration t, we update the model parameters as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{b}\sum_{i=1}^{b} \nabla_\theta \mathcal{L}\big(f_{\theta_t}(x_i^{tmp}(t), x_i^{egs}(t), x_i^{que}), y_i\big) \tag{4}$$

where η is the learning rate, $\mathcal{L}$ is the cross-entropy loss function, b is the batch size, $\mathcal{B} = \{(x_i, y_i)\}_{i=1}^{b}$ is the data batch, and y is the groundtruth label.

3) Inference. After the progressive fine-tuning, we have trained the LLMs with updated model parameters θ_T to perform inference without adding instructions or any examples. Thus, we can predict the output simply with $f_{\theta_T}(x^{que})$.

Our objective is to effectively compress and incorporate prompt knowledge into model parameters that are specifically tailored for distinct tasks. In pursuit of this goal, we have adopted PEFT during the fine-tuning phase of PromptIntern. Specifically, we apply LoRA (Hu et al., 2021) as it imposes no additional computational costs during inference and allows for scalable deployment across multiple tasks (Sheng et al., 2023). Note that our outlined pipeline in Algorithm 1 is also compatible with other PEFT techniques.

5 Experiment

In this section, we evaluate the performance of PromptIntern across various benchmarks on the NL2Code task, which is widely recognized for its utility in evaluating LLMs on both fine-tuning efficacy and cost-effectiveness in real-world applications (Zan et al., 2022). Our experiments primarily focus on two key perspectives: 1) **Effectiveness**: assessing the performance of textbf during inference phases; 2) **Efficiency**: quantifying the reduction in token usage and corresponding cost savings achievable through PromptIntern.

5.1 Settings

Datasets We apply three typical NL2Code datasets: MBPP (Austin et al., 2021) for NL to python code generalization, NL2F (Zhao et al., 2024) for NL to Excel spreadsheet formulas generation, NL2Bash (Lin et al., 2018) for NL to Bash Shell commands generation. Please refer to Appendix A.1 for the dataset details.

Evaluation Metrics We use one-shot pass accuracy $Pass@1$ (Austin et al., 2021) for MBPP, *Exact Match (E.M.)* for NL2F , and *BLEU* score for NL2Bash. In addition, we calculate the input token usage and compression rate τ for each dataset.

Table 1: Comparison with Prompt Compression Approaches

Methods	MBPP			NL2F			NL2Bash		
(*Inference on GPT-3.5*)	Pass@1	Tokens	$1/\tau_{all}$	E.M.	Tokens	$1/\tau_{all}$	BLEU	Tokens	$1/\tau_{all}$
GPT4 Generation	61.8	128	1.8x	59.6	425	1.6x	59.5	256	1.9x
Selective Context	59.7	102	2.2x	56.4	391	1.7x	55.2	158	3.1x
LLMLingua	70.3	115	2.0x	64.2	417	1.6x	61.3	154	3.1x
LongLLMLingua	65.2	121	1.9x	67.8	425	1.6x	58.4	133	3.6x
LLMLingua-2	72.5	107	2.1x	70.4	407	1.7x	62.8	141	3.4x
PromptIntern	78.1	107	2.1x	81.4	407	1.7x	70.5	141	3.4x

Baselines We consider two types of baselines with setups below:

1) *Prompt Compression approaches.* We employ the latest advancements in prompt compression techniques. Specifically, we utilize Gist Tokens (Mu et al., 2024), GPT-4 Generation (Jiang et al., 2023b), Selective Context(Li et al., 2023b), and LLMLingua series (Jiang et al., 2023a,b; Pan et al., 2024). Each prompt compression method is initially applied to compress the entire dataset to a predetermined compression rate. Then, the compressed dataset is utilized for both fine-tuning and inference evaluation.

2) *Direct Fine-tuning approaches.* We use "Direct" as the counterpart to our progressive fine-tuning strategy. Specifically, we adopt several conventional direct fine-tuning configurations, including i) direct fine-tuning with complete template and examples (e.g. *Template with 5-shots* in Table 2), ii) direct fine-tuning with compressed template and reduced examples (e.g. *Template x0.6 with 2-shots* in Table 2), iii) direct fine-tuning with template only (*Template only*), and iv) direct fine-tuning without template and examples (*No template*).

Models To demonstrate the broad applicability of PromptIntern, we utilize both closed-source and open-source LLMs with different parameter sizes for fine-tuning and inference processes.1) Closed-Source: We apply GPT-4-0613 (OpenAI, 2023), abbreviated as GPT-4, and GPT-3.5-turbo-0125[1], abbreviated as GPT-3.5. 2) Open-Source: We apply Mixtral-8x7B-v0.1 (Jiang et al., 2024), abbreviated as Mixtral-8x7B, Llama2-7B (Touvron et al., 2023), and Llama2-13B (Touvron et al., 2023).

Implementation Details Please refer to Appendix A for the additional experiments settings and implementation details.

[1]https://platform.openai.com/docs/models/gpt-3-5-turbo

5.2 Main results

Prompt Compression Approaches Comparison Table 1 reports the overall result of PromptIntern with the prompt compression baselines inferenced on GPT-3.5 across all datasets. Here we establish the template compression rate τ_{tmp} at 0.3 across all prompt compression approaches as well as PromptIntern to ensure a fair comparison. And τ_{all} in the table represents the overall prompt's compression rate. We observe that while utilizing a comparable number of tokens for inference, our approach significantly outperforms all baselines, achieving improvements of **5.6%** on MBPP, **11.0%** on NL2F, and **7.7%** on NL2Bash. The result demonstrates that PromptIntern generally offers the best balance of efficiency and effectiveness across varied tasks. Note that since the Gist Token(Mu et al., 2024) baseline is only applicable on open-source LLMs, we separately compare it with our approach which can be found at Appendix A.3.

Direct Fine-tuning Approaches Comparison Table 2, 3, and 4 demonstrate the comparison results of PromptIntern with different direct fine-tuning baselines. For the MBPP dataset, our approach significantly outperforms *No Template* and *Template* baselines by **9.6%-10.8%** and **0.6-1.3%**, respectively, and achieve comparable performance against *Template x0.6 with 2-shots* baseline with far less token usage. In addition, our approach achieves a range of **9.8x-12.2x** reduction in the number of input tokens required for comparable performance. Note that even though the *template with 5-shots* achieves the best performance, it requires **22.2x-27.4x** more input tokens than our approach. This underscores the efficiency of our method in reducing the computational resources required for inference, while still delivering robust performance.

Table 2: Comparison with direct fine-tuning baselines on MBPP datasets.

Model	Template *with* 5-shots		Template x0.6 *with* 2-shots		Template Only		No Template		**PromptIntern**	
	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens	Pass@1	Tokens
GPT-4	91.6	1181	87.4	424	87.3	226	77.2	43	87.9	43
GPT-3.5	82.7	1181	76.2	424	75.3	226	65.8	43	76.6	43
Mixtral-8x7B	69.8	1263	65.8	453	65.7	238	56.3	54	66.3	54
Llama2-13B	39.2	1286	37.5	471	36.4	251	26.4	58	37.1	58
Llama2-7B	30.4	1286	27.7	471	27.3	251	18.3	58	27.9	58

Table 3: Comparison with direct fine-tuning baselines on NL2F datasets.

Model	Template *with* 10-shots		Template x0.6 *with* 5-shots		Template Only		No Template		**PromptIntern**	
	E.M.	Tokens	E.M.	Tokens	E.M.	Tokens	E.M.	Tokens	E.M.	Tokens
GPT-4	94.8	3540	92.1	1838	89.7	680	82.5	286	91.6	286
GPT-3.5	85.5	3540	78.1	1838	76.2	680	70.4	286	78.4	286
Mixtral-8x7B	69.3	4204	66.3	2191	63.8	814	54.2	339	65.2	339
Llama2-13B	59.2	4202	54.9	2183	54.1	812	32.9	339	55.3	339
Llama2-7B	45.4	4202	40.7	2183	38.5	812	21.8	339	40.8	339

Table 4: Comparison with direct fine-tuning baselines on NL2Bash datasets.

Model	Template *with* 10-shots		Template x0.6 *with* 5-shots		Template Only		No Template		**PromptIntern**	
	BLEU	Tokens	BLEU	Tokens	BLEU	Tokens	BLEU	Tokens	BLEU	Tokens
GPT-4	86.7	1063	81.3	810	78.6	484	71.2	52	82.5	52
GPT-3.5	74.2	1063	67.5	810	65.1	484	61.2	52	67.7	52
Mixtral-8x7B	63.8	1320	58.3	1053	54.9	603	47.6	68	57.2	68
Llama2-13B	47.1	1244	43.9	988	41.6	574	35.1	64	43.5	64
Llama2-7B	35.8	1244	32.7	988	31.4	574	22.1	64	31.6	64

For the NL2F dataset, the results in Table 3 shows our approach greatly outperforms *No Template* baseline by **8.0%-19.0%** with the same token usage for inference. In addition, our approach achieves comparable performance to the *Template x0.6 with 2-shots* baseline while reducing the required number of input tokens by **6.4x**. Another finding from the result is that for LLMs with larger parameters, removing the prompt template causes less degradation in performance compared to smaller models, as seen in *Template Only* and *No Template* columns (-5.7% with LLama2-13B and -21.2% with LLama2-7B). This suggests that larger models have a better inherent capability to understand the input questions, even without detailed instructions provided by the prompt template.

For the NL2Bash dataset, we observe that under similar input tokens required, our approach also outperforms *No Template* baselines by **7.3%-11.3%**, showing the superiority of prompt internalization over direct fine-tuning. Moreover, for reaching a similar performance as *Template x0.6 with 5-shots*, our approach reduces the required tokens for inference by around **15.5x**. These results confirms the balance of cost efficiency and performance effectiveness of our PromptIntern approach during inference.

For additional experiments, please refer to Appendix A.4.

5.3 Ablation Study

To effectively assess the impact of various components within *our*, we introduce three variants of PromptIntern for ablation studies:

- **PromptIntern w/τ_{tmp}=0.3**, where we set the compression rate to 0.3 instead of 0 in template compression.
- **PromptIntern w/o Example Absorption**, in which we omit the example absorption for retrieving and internalizing few-shot examples during fine-tuning.
- **PromptIntern w/o Template Compression**, where template compression is excluded for both fine-tuning and inference prompt instances.

The overall results is shown in Table 5. When comparing PromptIntern with PromptIntern

Table 5: Ablation Study of PromptIntern.

Methods	MBPP			NL2F			NL2Bash		
(*Inference on GPT-3.5*)	Pass@1	Tokens	$1/\tau_{all}$	E.M.	Tokens	$1/\tau_{all}$	BLEU	Tokens	$1/\tau_{all}$
PromptIntern	76.6	43	5.3x	78.4	286	2.4x	67.7	52	9.3x
w/ τ_{tmp} = 0.3	78.1	107	2.1x	81.4	407	1.7x	70.5	241	2.0x
w/o Example Absorption	72.9	43	5.3x	73.5	286	2.4x	64.6	52	9.3x
w/o Template Compression	80.2	226	1.0x	83.6	680	1.0x	73.5	484	1.0x

w/τ_{tmp} $=$ 0.3, we observe an average of 2.4% drop on performance but a 3.7x compression on tokens across all three datasets. This highlights the balance between compression rate and accuracy performance. When comparing *our* with *our* w/o Example Absorption, we observe a significant performance drop in the latter variant, despite both approaches utilizing the same number of tokens for inference. This outcome highlights the importance of example absorption in internalizing essential information during the fine-tuning process. When comparing PromptIntern with PromptIntern w/o Template Compression, we note that adding the template compression saves an average of 280 tokens across the datasets but experiences an average performance drop of 5%. This demonstrates that while totally internalizing the template into model parameters significantly reduces token usage, it requires a trade-off in terms of inference performance.

Table 6: Comparison of schedule pattern and example retrival bank of PromptIntern. The results are inferenced on GPT-3.5.

PromptIntern	MBPP(Pass@1)	NL2F(E.M.)	NL2Bash(BLEU)
Pattern of Schedule $\mathcal{S}$			
- exp	74.8	72.5	59.4
- $\exp^{-1}$	67.3	64.9	52.8
- linear (**ours**)	77.6	78.4	67.7
Example Retrival Bank			
- 25%	75.9	77.5	66.2
- 50%	76.1	78.1	66.8
- 100% (**ours**)	77.6	78.4	67.7

5.4 Analysis on Schedule Pattern

In Table 6, we test the effectiveness of different scheduling patterns during the progressive fine-tuning process, specifically focusing on how the decreasing speed curve influences the compression of the template and absorption of few-shot examples. The patterns tested include exponential, inverse-exponential, and linear decrease.

From the data in the table, we observe that the linear decreasing schedule delivers the most consistent and highest performance across all three evaluation metrics, indicating superior performance in both parsing efficiency and language model understanding. Conversely, the inverse-exponential schedule shows the least effectiveness, with scores considerably lower in all metrics compared to the linear schedule. The exponential decrease performs moderately, but still falls short of the linear schedule, suggesting that a steady, predictable reduction is more beneficial than more aggressive decrease. This analysis suggests that for adopting a linearly decreasing schedule for progressive fine-tuning may lead to better performance in terms of accuracy compared to other scheduling patterns.

5.5 Analysis on Examples Retrieval Bank

Table 6 examines the impact of varying proportion of the training set used for constructing relevant examples in the examples retrieval bank. The options tested include using 25%, 50%, and 100% of the training set. The results clearly show a trend where increasing the percentage of the training set used in the examples retrieval bank correlates with improved performance. This suggests that larger examples retrieval bank provides a richer set of few-shots for the model to learn from, thereby enhancing its ability to generalize and perform accurately across tasks.

6 Conclusion

In this paper, we introduce PromptIntern, a novel method for prompt internalization that internalizes repetitive prompt knowledge into LLMs parameters. We develop specific compression strategies for different components of the prompt, accompanied by a tailored progressive fine-tuning pipeline. Experiments demonstrates that our method not only accelerates inference speed and reduces token usage but also maintains comparable performance effectiveness.

7 Limitations

While PromptIntern significantly reduces costs during the inference stage, the progressive fine-tuning approach incurs additional computational expenses during training. Specifically, our methodology demands substantial manual intervention for preprocessing and parameter adjustments throughout the fine-tuning process. Moreover, the current evaluation of our method is limited to a single task, specifically NL2Code. This restricts our understanding of its generalizability and effectiveness across a broader range of tasks. In future work, we plan to conduct extensive evaluations on more complex and varied tasks, such as long document summarization and question answering within specialized technical domains.

8 Ethics Statement

This research does not raise any ethical concerns. We obtained data only from publicly available sources where users have consented to the public sharing of their posts. We have conducted a thorough assessment to ensure that our research does not pose any potential harm.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Xiang Chen, Lei Li, Ningyu Zhang, Xiaozhuan Liang, Shumin Deng, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. 2022. Decoupling knowledge from memorization: Retrieval-augmented prompt learning. In *Advances in Neural Information Processing Systems*.

Zhuyun Dai, Vincent Y Zhao, Ji Ma, Yi Luan, Jianmo Ni, Jing Lu, Anton Bakalov, Kelvin Guu, Keith Hall, and Ming-Wei Chang. 2022. Promptagator: Few-shot dense retrieval from 8 examples. In *The Eleventh International Conference on Learning Representations*.

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

Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. *Advances in neural information processing systems*, 32.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. In *International Conference on Learning Representations*.

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, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Jonas Hübotter, Bhavya Sukhija, Lenart Treven, Yarden As, and Andreas Krause. 2024. Active few-shot fine-tuning. *arXiv preprint arXiv:2402.15441*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Huiqiang Jiang, Qianhui Wu, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023a. Llmlingua: Compressing prompts for accelerated inference of large language models. *arXiv preprint arXiv:2310.05736*.

Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023b. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839*.

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

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri

Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023a. Starcoder: may the source be with you! *Preprint*, arXiv:2305.06161.

Yucheng Li, Bo Dong, Chenghua Lin, and Frank Guerin. 2023b. Compressing context to enhance inference efficiency of large language models. *arXiv preprint arXiv:2310.06201*.

Xi Victoria Lin, Chenglong Wang, Luke Zettlemoyer, and Michael D Ernst. 2018. Nl2bash: A corpus and semantic parser for natural language interface to the linux operating system. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*.

Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023. Wizardcoder: Empowering code large language models with evol-instruct. *Preprint*, arXiv:2306.08568.

Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. 2023. Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. *arXiv preprint arXiv:2305.16938*.

Jesse Mu, Xiang Li, and Noah Goodman. 2024. Learning to compress prompts with gist tokens. *Advances in Neural Information Processing Systems*, 36.

Noor Nashid, Mifta Sintaha, and Ali Mesbah. 2023. Retrieval-based prompt selection for code-related few-shot learning. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 2450–2462. IEEE.

R OpenAI. 2023. Gpt-4 technical report. arxiv 2303.08774. *View in Article*, 2(5).

Zhuoshi Pan, Qianhui Wu, Huiqiang Jiang, Menglin Xia, Xufang Luo, Jue Zhang, Qingwei Lin, Victor Rühle, Yuqing Yang, Chin-Yew Lin, et al. 2024. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression. *arXiv preprint arXiv:2403.12968*.

Anastasiia Razdaibiedina, Yuning Mao, Madian Khabsa, Mike Lewis, Rui Hou, Jimmy Ba, and Amjad Almahairi. 2023. Residual prompt tuning: improving prompt tuning with residual reparameterization. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 6740–6757.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Ohad Rubin, Jonathan Herzig, and Jonathan Berant. 2022. Learning to retrieve prompts for in-context learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2655–2671.

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*.

Jake Snell, Kevin Swersky, and Richard Zemel. 2017. Prototypical networks for few-shot learning. *Advances in neural information processing systems*, 30.

Yuan Sui, Jiaru Zou, Mengyu Zhou, Xinyi He, Lun Du, Shi Han, and Dongmei Zhang. 2023. Tap4llm: Table provider on sampling, augmenting, and packing semi-structured data for large language model reasoning. *arXiv preprint arXiv:2312.09039*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

David Wingate, Mohammad Shoeybi, and Taylor Sorensen. 2022. Prompt compression and contrastive conditioning for controllability and toxicity reduction in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5621–5634.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2023. Recomp: Improving retrieval-augmented lms with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*.

Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Yongji Wang, and Jian-Guang Lou. 2022. Large language models meet nl2code: A survey. *arXiv preprint arXiv:2212.09420*.

Haoyu Zhang, Jianjun Xu, and Ji Wang. 2019. Pretraining-based natural language generation for text summarization. *arXiv preprint arXiv:1902.09243*.

Wei Zhao, Zhitao Hou, Siyuan Wu, Yan Gao, Haoyu Dong, Yao Wan, Hongyu Zhang, Yulei Sui, and Haidong Zhang. 2024. Nl2formula: Generating spreadsheet formulas from natural language queries. *arXiv preprint arXiv:2402.14853*.

Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. 2023. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*.

A Additional Experiments

A.1 Dataset Details

MBPP The MBPP dataset, as detailed by (Mosbach et al., 2023), consists of Python programming tasks, each accompanied by a description in natural language that has been expertly curated. The dataset is segmented into training and test sets, with 974 and 102 examples, respectively.

NL2F The NL2F dataset, as detailed by (Zhao et al., 2024), consists of 70,799 pairs of NL queries and spreadsheet formulas and covers 21,670 tables. We follow the dataset instructions (Zhao et al., 2024) to randomly split data into a training set (75%), validation set (10%), and test set (15%).

NL2Bash The nl2bash dataset, as described by (Lin et al., 2018), comprises snippets of Bash code, each paired with a natural language description expertly curated. The dataset is divided into training and test sets, containing 8,090 and 606 examples, respectively.

A.2 Implementation Details

Fine-tuning Procedures For PromptIntern training, we adopt LoRA (Hu et al., 2021) with a rank of 32. For GPT-series and open-source model fine-tuning we train models for MBPP/NL2F/NL2Bash with 6/12/12 epochs, 16/128/128 batch size, 200/200/200 checkpoint interval, and 4096/4096/4096 context window length , respectively.

Model Inference We provide the detailed parameters we adopted during fine-tuned LLM inference: temperature equal to 0, max tokens equal to 1028, top p equal to 0.95, presence penalty equal to 0, and frequency penalty equal to 0

Baseline Settings For prompt compression baselines comparison, we set the template compression ratio $\tau_{tmp} = 0.3$. For direct fine-tuning baselines, we apply LLMLingua-2 (Pan et al., 2024) as the default template compressor as it performs the best in Table 1.

PromptIntern Settings Number of top-k: We set the initial k as 5/10/10 across MBPP/NL2F/NL2Bash for the initial number of few-shot examples for example absorption. During progressive fine-tuning, we decrease k linearly in the order of 5-2-0/10-5-0/10-5-0 across MBPP/NL2F/NL2Bash. Number of τ_{tmp}: For the prompt compression baseline experiments, we set the final template rate to 0.3, which is used in the last stage of fine-tuning as well as inference. For the other experiments and ablation studies, we set the final template rate to 0 to achieve fully internalization.

Cost Evaluation We compute the total costs based on the price shown in OpenAI Pricing[2]

Computational Resource We conduct all experiments on one A100x1-80G computational cluster.

A.3 Comparison with Gist Tokens

We report the comparison result of PromptIntern with Gist Tokens (Mu et al., 2024) on Table 9. Gist Tokens showcases consistent performance, with notable results in NL2Bash where it achieves a BLEU score of 22.7, suggesting a moderate alignment with the dataset's requirements. In contrast, PromptIntern demonstrates superior performance

[2]https://openai.com/api/pricing/

Table 7: Speed (s/instance) Comparison of PromptIntern with Direct Fine-tuning baseline on MBPP dataset.

Model	Template *with* 5-shots	Template x0.6 *with* 2-shots	Template	No template	**PromptIntern**
GPT-4	10.21	8.68	7.29	4.36	4.17
GPT-3.5	5.43	3.68	3.06	1.35	1.31
Mixtral-8x7B	4.84	3.23	3.14	1.76	1.62
Llama2-13B	3.17	2.54	2.19	1.08	1.13
Llama2-7B	2.95	2.27	1.95	0.84	0.76

Table 8: Speed (s/instance) Comparison of PromptIntern with Direct Fine-tuning baseline on NL2F dataset.

Model	Template *with* 10-shots	Template x0.6 *with* 5-shots	Template	No template	**PromptIntern**
GPT-4	12.47	8.43	4.16	2.12	2.15
GPT-3.5	8.16	5.26	2.18	1.46	1.44
Mixtral-8x7B	6.27	4.71	3.17	1.19	1.2
Llama2-13B	4.15	2.95	1.25	0.63	0.63
Llama2-7B	3.83	2.03	1.24	0.41	0.39

Table 9: Comparison with Gist Tokens (Mu et al., 2024)

Methods	MBPP			NL2F			NL2Bash		
(*Inference on Llama2-7B*)	Pass@1	Tokens	$1/\tau_{all}$	E.M.	Tokens	$1/\tau_{all}$	BLEU	Tokens	$1/\tau_{all}$
Gist Tokens	10.2	61	4.1x	17.5	342	2.4x	22.7	66	8.6x
PromptIntern	27.9	58	4.3x	40.8	339	2.4x	31.6	64	9.0x

across all metrics and datasets, particularly excelling in the NL2Bash dataset with a BLEU score of 31.6 and maintaining similar efficiency in token usage. The results demonstrate the our approach significantly outperforms the Gist token while conducting overall the same compression rate.

A.4 Comparison on Inference Speed

The experimental results presented in Tables 7 and 8 illustrate the low latency characteristics of PromptIntern during inference across two datasets, MBPP and NL2F. Specifically, for the MBPP dataset, PromptIntern achieves an inference speed of 4.17 instances per second on the GPT-4 model, closely aligning with the 4.36 instances/s observed in the no template setup and far surpassing the more resource-intensive template with 5-shots configuration at 10.21 instances/s. In the NL2F dataset, PromptIntern similarly demonstrates its efficiency with an inference speed of 2.15 instances/s for GPT-4, which is nearly equivalent to the 2.12 instances/s observed without any template and significantly outperforms the elaborate template with 10-shots configuration, which achieves 12.47 instances/s. These results highlight PromptIntern's capability to maintain competitive inference speeds while minimizing latency efficienlty.

B Example Demonstration

We demonstrate an an example on how we preprocess an input prompt through both template compression and example absorption in Figure 3

C Prompts

Please refer to Figure 4,5, and 6 for the detailed prompts.

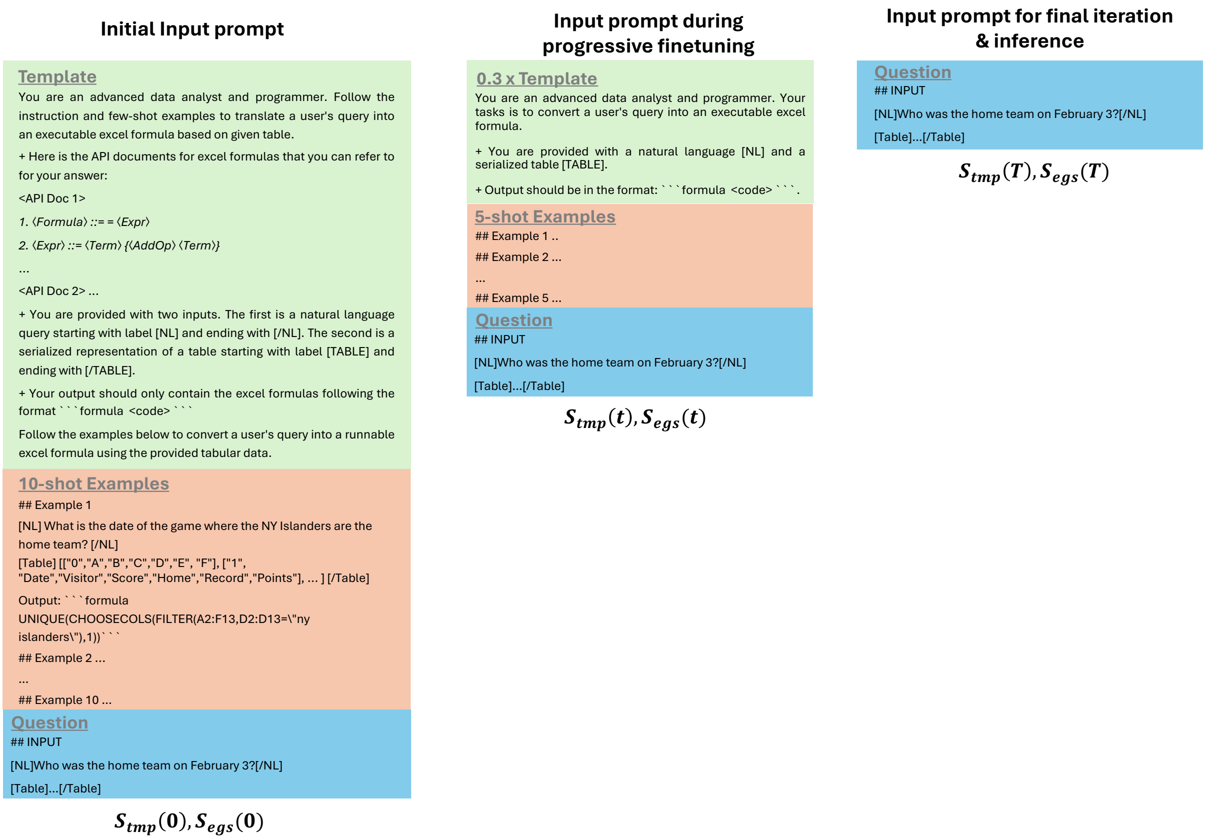

Figure 3: An Example from NL2F demonstrating how an original prompt is preprocessed through template compression and example absorption in PromptIntern for progressive finetuning and final inference.

MBPP Generation Prompt

[Template]
You are an advanced Python programmer.
Read the instructions claimed below and write the corresponding Python code.
You will be given a question describing the python function need to implement for.
You will also be given three corresponding test cases written in Python code. They all using assert styles.
Read the question and test cases carefully and fulfill the requirements below:
+ Your written function's name should be the same as the function name shown in the test cases.
+ Your function should take the same number of input arguments and output values as shown in the test cases.
+ Your function should handle same type of input and return the same type of value as shown in the test cases.
+ Your function should pass all the three provided test cases.
+ You can use any built-in python libraries.
+ Your output should strictly follow the format of ```python <code>```.

[Example]
Example 1
...
Example 2
...

[Question]
NL Question: ...
Three Test Cases: ...

Figure 4: Prompts of MBPP

NL2F Generation Prompt

[Template]
You are an advanced data analyst and programmer. Follow the instruction, referred API documents, and few-shot examples to translate a user's query into an executable excel formula based on the given table.
+ Here is the API documents for excel formulas that you can refer to for your answer:
<API Doc 1>
<API Doc 2>
<API Doc 3>
...
+ You are provided with two inputs. The first is a natural language query starting with label [NL] and ending with [/NL]. The second is a serialized representation of a table starting with label [TABLE] and ending with [/TABLE].
+ Your output should only contain the excel formulas following the format ` ` `formula <code> ` ` `
Follow the examples below to convert a user's query into a runnable excel formula using the provided tabular data.

[Example]
Example 1
...
Example 2
...

[Question]
[NL] ... [/NL]
[TABLE] ... [/TABLE]

Figure 5: Prompts of NL2F

NL2Bash Generation Prompt

[Template]
You are an advanced shell programmer. Follow the instruction, referred API documents, and few-shot examples to translate a user's natural language command into an executable Bash command.

+ Here is the API documents for advanced bash shell functions and commands that you can refer to for your answer:
<API Doc 1>
<API Doc 2>
<API Doc 3>
...
+ You are provided with one input. The first is a natural language query starting with label [NL] and ending with [/NL].
+ Your output should only contain the excel formulas following the format ` ` `bash <code> ` ` `

Follow the examples below to convert a user's query into a runnable bash command.

[Example]
Example 1
...
Example 2
...

[Question]
[NL] ... [/NL]

Figure 6: Prompts of NL2Bash