Enhancing Reasoning Capabilities of Small Language Models with Blueprints and Prompt Template Search

Dongge Han[1] **Menglin Xia**[1] **Daniel Madrigal Diaz**[1] **Samuel Kessler**[1] **Ankur Mallick**[1] **Xuchao Zhang**[1]
Mirian Del Carmen Hipolito Garcia[1] **Jin Xu**[1] **Victor Rühle**[1] **Saravan Rajmohan**[1]

Abstract

Small language models (SLMs) offer promising and efficient alternatives to large language models (LLMs). However, SLMs' limited capacity restricts their reasoning capabilities and makes them sensitive to prompt variations. To address these challenges, we propose a novel framework that enhances SLM reasoning capabilities through LLM generated blueprints. The blueprints provide structured, high-level reasoning guides that help SLMs systematically tackle related problems. Furthermore, our framework integrates a prompt template search mechanism to mitigate the SLMs' sensitivity to prompt variations. Our framework demonstrates improved SLM performance across various tasks, including math (GSM8K), coding (MBPP), and logic reasoning (BBH). Our approach improves the reasoning capabilities of SLMs without increasing model size or requiring additional training, offering a lightweight and deployment-friendly solution for on-device or resource-constrained environments.

1. Introduction

Large language models (LLMs) have demonstrated remarkable capabilities across a wide range of real-world applications. However, deploying LLMs often requires substantial computational resources, posing challenges in efficiency and their general applicability (Wan et al., 2023; Chowdhery et al., 2023; Du et al., 2022). Small language models (SLMs) (OpenAI, 2024b; Jiang et al., 2023) such as Phi3-mini with only 3.8B parameters (Abdin et al., 2024), offer promising alternatives by providing efficient solutions that are also more accessible in resource-constrained environ-

ments.

Despite these advantages, SLMs still face challenges that hinder their broader adoption. Due to their limited capacity, SLMs exhibit less reasoning capabilities such as in chain-of-thought (CoT) and in-context learning, as reported by (Magister et al., 2022; Wei et al., 2023; Brown, 2020). For example, unlike LLMs, which can effectively extract abstract chain-of-thought reasoning patterns from the in-context examples and apply to new problems, SLMs' limited reasoning capacity sometimes make it challenging for them to effectively extract and transfer insights from the examples to new problems, limiting their generalization capabilities.

To address these challenges, we propose a novel framework that enhances SLM problem-solving via *blueprints*. A blueprint is a reusable, step-by-step reasoning guide designed to help SLMs solve a class of related problems. The blueprint is generated with an LLM by extracting high-level, abstract reasoning instructions from concrete examples, facilitating SLMs with a clear and actionable plan for tackling complex problems. Unlike CoT or in-context learning, where SLMs may struggle to come up with, or extract from the examples correct reasoning steps and generalize to new problems, blueprints provide explicit reasoning guidance to bridge this gap. By outlining high-level reasoning steps and sometimes incorporating illustrative examples, blueprints facilitate SLMs with the abstract reasoning steps and how to apply them to solve new problems. Moreover, we tailor the blueprint to each problem class and SLM by selecting from diverse blueprint styles and refining it using Automatic Prompt Optimization (APO) (Pryzant et al., 2023). In addition to blueprint generation, our framework incorporates a prompt template search mechanism as a further contribution to enhance the consistency of SLM performance, addressing their sensitivity to prompt variations.

We evaluate our approach on various SLMs - GPT4o-mini (OpenAI, 2024b), Mistral-7B (Jiang et al., 2023) and Phi3-mini, across a diverse set of tasks, including math (GSM8K Cobbe et al. (2021)), coding (MBPP Austin et al. (2021)), and logic reasoning (BBH Suzgun et al. (2022)). The experimental results demonstrate that our framework
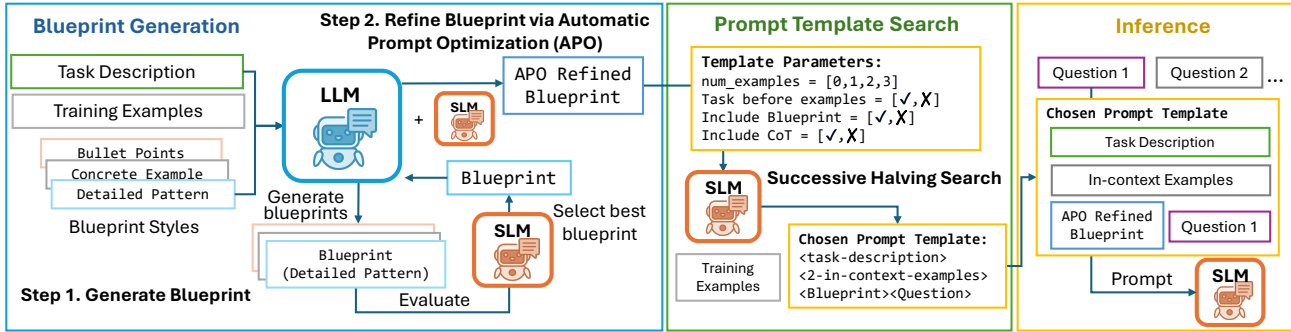
[1]Microsoft. Correspondence to: Dongge Han <dongge-han@microsoft.com>.

Figure 1. Illustration of our framework. During training, for each SLM and task category, an LLM is used to generate blueprints using various input styles, from which the best blueprint is picked, and refined via APO. Then we perform template search to find the best prompt template. During inference, the chosen blueprint and template are reused across problems within the task category to form the input prompt to the SLM.

Figure 2. Example LLM generated blueprint. The blueprint includes high-level reasoning steps, and optionally an in-context example question and solution process as a guide for how to apply the reasoning steps.

consistently improves SLM performance, outperforming state-of-the-art methods. These findings highlight the effectiveness of blueprint-guided reasoning and prompt template optimization in overcoming the inherent limitations in the reasoning capabilities and sensitivity of SLMs.

2. Related Work

Prompt engineering is an efficient way to enhance LLMs, including methods like CoT (Wei et al., 2022), ReAct (Yao et al., 2022), ToT (Long, 2023), and few-shot prompting (Brown, 2020), as well as automatic prompt engineering methods (Lester et al., 2021; Deng et al., 2022; Ma et al., 2024; Yang et al., 2024) such as APO (Pryzant et al., 2023), APE (Zhou et al., 2022), PromptBreeder (Fernando et al., 2023), and PromptWizard (Agarwal et al., 2024). These methods primarily focus on refining a short, high-level task description for an LLM. In contrast, our framework specifically targets the unique challenges of SLMs, such as limited

reasoning capacity and prompt sensitivity, by generating reusable and structured blueprints from example problems and prompt template optimization. These blueprints provide explicit multi-step guidance tailored to SLM needs, offering detailed clarification and actionable reasoning support. While we used APO for blueprint refinement in our experiments, our framework is also versatile to integrate other aforementioned prompt refinement methods.

3. Method

In this section, we present our framework designed to enhance the reasoning capability of SLMs with blueprints and prompt template search.

3.1. Problem Formulation

Fig. 1 shows an overview of our framework. We refer to a **task** $\mathcal{T}$ as a collection of related **problems** (e.g., a math question) within the same category (e.g., causal judgement, math reasoning). During training, for each task category $\mathcal{T}$ and SLM model $\mathcal{M}_{slm}$ (e.g., Phi3-mini), we use a small set of examples to generate a **blueprint** that is tailored to guide the SLM with problem-solving for problems within $\mathcal{T}$; we also generate a **prompt template** (e.g., `<task-description><2-in-context-examples><blueprint><question>`, which defines the prompt components and their ordering). During inference, the blueprint and prompt template can be efficiently reused to form prompts to $\mathcal{M}_{slm}$ across incoming problems within task $\mathcal{T}$.

3.2. Blueprint Generation and Optimization

To address the limited reasoning capabilities of SLMs, we leverage an LLM $\mathcal{M}_{llm}$ (we use GPT4o (OpenAI, 2024a) in this work) to generate a reusable blueprint tailored per SLM per task. The blueprint serves as a structured guide,
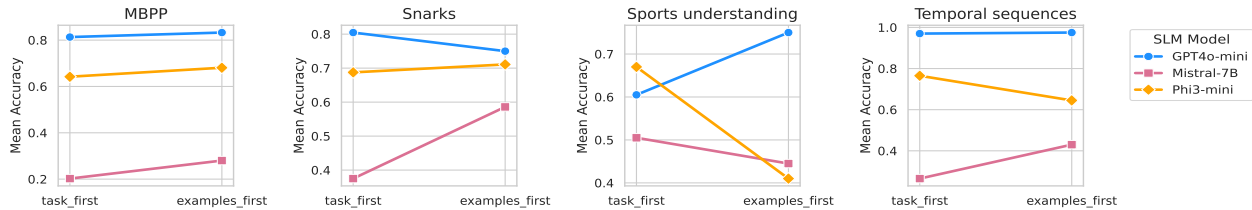
Figure 3. Sensitivity of SLMs to prompt variations across tasks - Even simple changes in the template, like swapping the order of `<task-description>` and `<in-context-example>`, can significantly affect performance, with impacts varying per SLM and task, e.g., in Sports understanding (Snarks, Sports understanding and Temporal sequences are sub-categories from the BBH dataset), GPT4o-mini performed better with examples before the task description, while Mistral-7B and Phi3-mini preferred the opposite. There is no universal best prompt template for all tasks and SLMs, making task and model-specific prompt optimization necessary.

outlining the general pattern of steps the SLM should follow to arrive at the correct solution, which can be reused across problems within the task category.

As shown in Fig. 1, we first generate the blueprint, then refine it via automatic prompt optimization (APO). An example generated blueprint is shown in Appendix Fig. 2. To generate the blueprint, we begin by sampling M example problems from $D_{train}^{\mathcal{T}}$, each consisting of a question and its step-by-step solution. These examples are concatenated into a single prompt to $\mathcal{M}_{llm}$ to generate the initial blueprint that encapsulates the shared reasoning pattern across the example problems. Since different SLMs may prefer different prompt *styles* (Fernando et al., 2023; Agarwal et al., 2024) depending on the task, we personalize the blueprint for each SLM by preparing K distinct blueprint generation styles $S = \{S_1, ...S_K\}$. These styles, such as `concise-highlevel`, `bullet-points`, vary in aspects such as the level of detail and formatting. To identify $\mathcal{M}_{slm}$'s preferred blueprint style on task $\mathcal{T}$, we augment the blueprint to the $\mathcal{M}_{slm}$ prompt and compare its performance on sampled training examples. The blueprint which yields the highest performance is selected. More details about the styles and blueprint generation prompts are included in Appendix A.

To further refine the blueprint according to $\mathcal{M}_{slm}$'s behavior on task $\mathcal{T}$, we employ Automatic Prompt Optimization (APO) (Pryzant et al., 2023). APO iteratively improves the blueprint by evaluating $\mathcal{M}_{slm}$ on training examples and identifying errors, which are compiled into an error message. The LLM generates error analyses from this message to update and paraphrase the blueprint, which are then re-evaluated, with the best-performing version selected for the next round of refinement. More details on the APO refinement procedure and prompts used are included in Appendix A.2.

3.3. Prompt Template Search

SLMs are often sensitive to the prompt format, such as the template and style variations. As shown in Fig. 3, we observed that simply interchanging the task description and in-context examples have a large effect on SLM performance. Furthermore, the effects of prompt format changes vary across different SLMs and tasks, indicating that there's not a universally optimal prompt template for all SLMs and tasks.

Motivated by this sensitivity of SLMs to prompt templates and the variability across different SLMs and tasks, we perform prompt template search to systematically identify the optimal prompt template for each SLM and task, as shown in Fig. 1. To begin with, we define a search space encompassing key prompt template parameters such as whether or not to include a blueprint. To efficiently search the space, We adopt a simplified successive halving search (Pryzant et al., 2023): we begin with evaluating all N possible templates on k sampled training examples, then iteratively narrow down the search space with a reduction factor f (e.g., $f = 2$), i.e., retain the $\lfloor \frac{N}{f} \rfloor$ top scoring candidates and re-evaluate, until one prompt template remains. By progressively narrowing down the template candidates, successive halving efficiently identifies the top prompt template for each SLM and task.

4. Experiments

In the experiments, we demonstrate the effectiveness of our proposed framework across various reasoning tasks and SLM models. Followed by the main findings, we present a qualitative example to illustrate the blueprint following behavior, to understand how the blueprint helps guide the SLM. Finally, we present an interesting finding about the different preferences of SLMs to the various blueprint styles.

		BBH			GSM8K			MBPP		
		GPT4o-mini	Mistral7B	Phi3-mini	GPT4o-mini	Mistral7B	Phi3-mini	GPT4o-mini	Mistral7B	Phi3-mini
Baseline	CoT (1-shot)	0.839	0.449	0.678	0.940	0.400	0.807	0.813	0.202	0.642
	CoT (3-shot)	0.850	0.53	0.686	0.923	0.347	0.780	0.821	0.233	0.658
	APO	0.825	0.435	0.651	0.930	0.407	0.797	**0.837**	0.288	0.658
Ours	BP (w.o. APO)	0.882	0.539	**0.743**	0.947	0.450	0.817	0.821	**0.440**	0.689
	BP (w. APO)	**0.884**	0.559	0.738	0.943	0.480	**0.840**	0.829	0.401	0.681
	BP (w. APO) + TS	**0.884**	**0.572**	0.722	**0.953**	**0.490**	0.820	0.833	0.424	**0.696**

Table 1. Performance comparison of different methods (rows) across various SLMs and datasets (GSM8K, MBPP, BBH). The table highlights the accuracy improvements of blueprint-based methods (BP) over CoT and APO (with task description) baselines. BP (w. APO) and BP (w.o. APO) refer to blueprint generation with/without further APO refinement, respectively. Both methods consistently outperform the CoT and APO baselines. The BP (w. APO) + TS (Template Search) variant demonstrates the best overall performance, showing optimal task accuracy in 5 out of 9 SLM/dataset combinations and near-optimal performance in the remaining ones.

4.1. Setup

We evaluate the effectiveness of our proposed framework for enhancing SLM reasoning across various reasoning tasks, using three different SLM models GPT4o-mini, Phi3-mini, and Mistral-7B on three widely used different domains:

- GSM8K (Mathematical reasoning)

- MBPP (Python coding)

- BBH task suite (Logic reasoning, comprising over 20 different task categories)

We measured the SLMs' solution accuracy on all datasets. For GSM8K and BBH, we measure the accuracy of the SLM answer compared with ground truth answer. For MBPP, the SLM generated code is correct if it passes all test cases. The compared methods are: 1. CoT (1-shot) and 2. CoT (3-shot): chain-of-thought prompting including "Let's think step by step", and 1 (or 3) in-context learning examples; 3. APO: baseline APO method which refines the task description, and 3 variants of our method: 4. BP (w.o. APO): where the SLM uses the top scoring blueprint style without further APO. (BP refers to blueprint); 5. BP (w. APO): using the blueprint refined with APO. Both variants use 1-shot example and no CoT; 6. BP (w. APO) + Template Search, using the APO refined blueprint and prompt template found via template search. For fair comparison, we use the same in-context examples and held-out test set (200 - 300 datapoints for GSM8K, MBPP and each BBH sub-category) across all compared methods. More details on experiment setup can be found in Appendix E

4.2. Main Results

Table 1 summarizes the performance comparison of the methods for different SLMs across GSM8K, MBPP and BBH. In comparison, our method BP (w.o. APO) already outperforms both CoT and APO baselines across almost

all datasets and SLMs, e.g., compared with CoT (3-shot), improving accuracy of $+20\%$ for Mistral-7B on MBPP, $+5.7\%$ for Phi3-mini on BBH, and $+3.2\%$ for GPT4o-mini on BBH. With APO, the BP (w. APO) variant slightly improves performance over the non-APO variant in only some categories, e.g., $+2\%$ for Mistral-7B on BBH, and $+3\%$ on GSM8K. The BP (w. APO) + Template Search variant shows best performance across all variants, outperforming all other methods in 5 out of 9 SLM/task combinations, and competitive performance in the rest. Note that the template search variant does not always provide the best result due to the small number of examples used to compare the template candidates during template search. These results demonstrate the effectiveness of our proposed framework across a wide range of reasoning tasks and various SLM models.

4.3. Example Blueprint and SLM Response following Blueprint Steps

To understand the format of the generated blueprint and how the SLM follows the blueprint, we show a qualitative example in Fig. 4. In the figure, we show an example blueprint generated for the logical reasoning tasks for geometric shapes reasoning category (from BBH) for GPT4o-mini, and example responses from GPT4o-mini before and after following the blueprint, respectively. As shown in the figure, the blueprint starts by listing the reasoning steps, including first defining the goal, then break down the information provided, clarifying the goal and decision criteria, and detailed step-by-step solutions, followed by a conclusion. Each high-level step is illustrated with a description of how to carry out the step, helping the SLM to clearly follow the instruction. After presenting the high-level steps, the blueprint also shows an example application, which guides the SLM on how to apply the blueprint to solve a concrete example problem.

To see the effectiveness of the blueprint, we show the response from GPT4o-mini with (in the green box) and with-

BLUEPRINT:
Step 1: Define the Goal - **Goal**: Identify the geometric shape drawn by the given SVG path element.
Step 2: Break Down the Information Provided - **SVG Path Element**: Analyze the commands and coordinates in the SVG path. - **Commands**: Identify the types of commands used (e.g., `M`, `L`, `A`). - **Coordinates**: Extract the coordinates associated with each command.
Step 3: Clarify the Goal and Decision Criteria - **Goal Clarification**: Determine the type of geometric shape based on the path commands and coordinates. - **Decision Criteria**: - **Number of Sides**: Count the number of line segments to determine the number of sides. - **Shape Properties**: Use properties such as side lengths, angles, and parallelism to distinguish between shapes with the same number of sides. - **Special Commands**: Recognize special commands like `A` for arcs to identify shapes like circles and ellipses.
Step 4: Step-by-Step Solution
1.   **Identify Commands and Coordinates**: - Parse the SVG path to list all commands and their associated coordinates.
2.   **Classify the Shape**: - **Count Line Segments**: Count the number of `L` commands to determine the number of sides. - **Analyze Properties**: For shapes with the same number of sides, analyze side lengths and angles to distinguish between them. - **Special Shapes**: Identify special shapes like circles and ellipses using the `A` command.
3.   **Match with Options**: - Compare the identified shape with the given options to select the correct answer.
Example Application:
1.   **Identify Commands and Coordinates**: - Example: `<path d="M 64.00,63.00 L 44.00,63.00 L 44.00,50.00 L 64.00,50.00 L 64.00,45.00 L 85.00,57.00 L 64.00,68.00 L 64.00,63.00"/>` - Commands: `M`, `L`, `L`, `L`, `L`, `L`, `L`, `L` - Coordinates: `(64.00,63.00)`, `(44.00,63.00)`, `(44.00,50.00)`, `(64.00,50.00)`, `(64.00,45.00)`, `(85.00,57.00)`, `(64.00,68.00)`, `(64.00,63.00)`
2.   **Classify the Shape**: - **Count Line Segments**: 7 `L` commands indicate a seven-sided shape. - **Analyze Properties**: No need for further analysis as "heptagon" is the only seven-sided option.
3.   **Match with Options**: - The shape is a heptagon. - **Answer**: (B) heptagon
Conclusion - Follow the blueprint to systematically identify the geometric shape drawn by the SVG path element. - Ensure each step aligns with the defined goal and decision criteria for accurate identification.

EXAMPLE SLM RESPONSE WITHOUT BLUEPRINT (GPT4o-mini): Let's think step by step. This SVG path element contains "M" and "L" commands. The "M" command moves the current point to the specified coordinates, while the "L" command draws a line from the previous point to the new coordinates. This path can be decomposed into several commands: 1. **M 15.44,15.80**: Move to (15.44, 15.80). 2. **L 22.07,7.46**: Draw a line to (22.07, 7.46). 3. **M 22.07,7.46**: Move to (22.07, 7.46) again (this is a new starting point). 4. **L 33.15,7.17**: Draw a line to (33.15, 7.17). 5. **L 43.50,16.57**: Draw a line to (43.50, 16.57). 6. **M 43.50,16.57**: Move to (43.50, 16.57). 7. **L 41.48,29.32**: Draw a line to (41.48, 29.32). 8. **M 41.48,29.32**: Move to (41.48, 29.32). 9. **L 36.23,36.66**: Draw a line to (36.23, 36.66). 10. **M 36.23,36.66**: Move to (36.23, 36.66). 11. **L 21.22,33.46**: Draw a line to (21.22, 33.46). 12. **L 15.44,15.80**: Draw a line back to the starting point (15.44, 15.80). Now, let's analyze the shape: - The path starts at (15.44, 15.80) and ends back at the same point, indicating a closed shape. - The path consists of multiple lines, but it also has several "M" commands, which indicate that the drawing is not a simple polygon but rather a more complex shape. To determine the type of shape: - The path does not create a simple polygon with a specific number of sides (like a triangle, rectangle, pentagon, etc.) due to the multiple moves and lines. - The shape appears to be more complex and does not fit neatly into the categories of regular polygons. Given the options: (A) circle (B) heptagon (C) hexagon (D) kite (E) line (F) octagon (G) pentagon (H) rectangle (I) sector (J) triangle The best fit for this complex shape, which does not conform to a simple polygon, is likely to be a **kite** due to its irregularity and the way it connects back to the starting point. So the answer is (D).

EXAMPLE SLM RESPONSE WITH BLUEPRINT (GPT4o-mini): To identify the geometric shape drawn by the given SVG path element `<path d="M 15.44,15.80 L 22.07,7.46 M 22.07,7.46 L 33.15,7.17 L 43.50,16.57 M 43.50,16.57 L 41.48,29.32 M 41.48,29.32 L 36.23,36.66 M 36.23,36.66 L 21.22,33.46 L 15.44,15.80"/>`, let's follow the blueprint step by step.
Step 1: Define the Goal - **Goal**: Identify the geometric shape drawn by the SVG path.
Step 2: Break Down the Information Provided - **SVG Path Element**: The path contains multiple commands. - **Commands**: `M` (move to), `L` (line to). - **Coordinates**: - Start at `(15.44, 15.80)`. - Line to `(22.07, 7.46)`. - Line to `(33.15, 7.17)`. - Line to `(43.50, 16.57)`. - Line to `(41.48, 29.32)`. - Line to `(36.23, 36.66)`. - Line to `(21.22, 33.46)`. - Finally, return to `(15.44, 15.80)`.
Step 3: Clarify the Goal and Decision Criteria - **Goal Clarification**: Determine the type of geometric shape based on the path commands and coordinates. - **Decision Criteria**: - **Number of Sides**: Count the number of line segments to determine the number of sides. - **Shape Properties**: Analyze properties such as side lengths and angles.
Step 4: Step-by-Step Solution 1. **Identify Commands and Coordinates**: - The path consists of 7 line segments (from the `L` commands). - The path starts and ends at the same point, indicating a closed shape. 2. **Classify the Shape**: - **Count Line Segments**: There are 6 `L` commands, which means there are 7 vertices (including the starting point). - **Analyze Properties**: The shape is closed and has 7 sides, which corresponds to a heptagon. 3. **Match with Options**: - The shape is a heptagon. - **Answer**: (B) heptagon
Conclusion Following the blueprint, the identified geometric shape drawn by the SVG path element is a heptagon. The answer is (B).

Figure 4. Example blueprint from the BBH geometric shape reasoning task, and example GPT4o-mini responses without and with blueprint, respectively. We can observe that the blueprint listed the reasoning steps, as well as showing an example application, which guides the SLM how to apply the blueprint to example problems. The SLM with blueprint indeed followed the reasoning steps as demonstrated in the blueprint. For each high-level step, the SLM first stated the high-level step provided by the blueprint, followed by its application of the abstract step to the concrete problem.

out (in the orange box) the blueprint. Note that without using the blueprint, the SLM answered the problem incorrectly even with chain-of-thought reasoning, while the response from GPT4o-mini using the blueprint answered the problem correctly. Comparing the answers, the response without blueprint directly dives into the detailed problem-solving steps, while the response following the blueprint presents better structured reasoning. We can observe that the SLM indeed followed the reasoning steps as demonstrated in the blueprint. Instead of diving into the detailed problem solution directly, the SLM starts by defining the problem goal, breaks down information gathered, then performs step-by-step solution.

4.4. SLM performance with different blueprint styles

To explore the underlying preferences of different SLMs with various blueprint styles, Fig. 5 presents the blueprint styles alongside the average SLM performance when using the blueprints, averaged over all task categories (i.e., 10 examples per blueprint style per category, for a total of 28 categories, including GSM8K, MBPP and all BBH sub-categories).

Overall, GPT4o-mini is demonstrated the highest level of robustness to the blueprint styles, with 3% performance difference between the best and worst performing styles, while Phi3-mini and Mistral-7B are slightly more sensitive, showing a 11% and 12% performance differences, respectively.

Interestingly, each SLM has different preferences for the blueprint styles – GPT4o-mini slightly prefers instructions
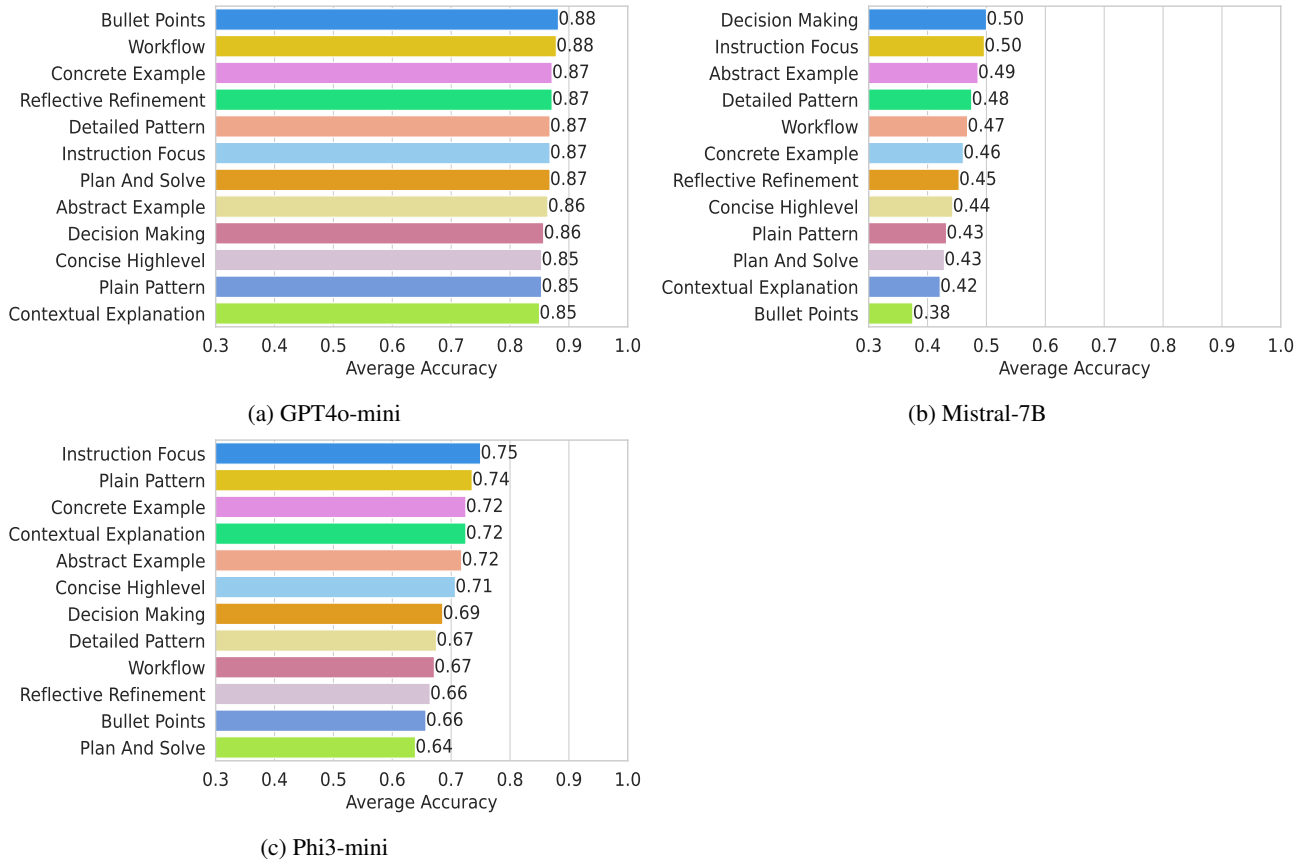
(a) GPT4o-mini

(b) Mistral-7B

(c) Phi3-mini

Figure 5. Accuracy of SLM task performance across blueprint styles. For each blueprint style and SLM, the performance is averaged over 280 examples (10 samples per task category for a total of 28 task categories spanning the GSM8K, MBPP and BBH datasets.)

in bullet pointed format and illustration with workflows or concrete examples, while it was slightly less effective with `contextual-explanation`, `plain-pattern` and `concise-highlevel`. In contrast, Phi3-mini preferred `instruction-focus`, `plain-pattern` and `concrete-example`, which put stronger focus on clear and actionable guidance, while it was less effective with the multi-step styles such as `reflective-refinement` and `plan-and-solve`, and `bullet-points` formatted instructions. Finally, Mistral-7B preferred blueprint styles with `decision-making`, `instruction-focus` and `abstract-example`, which provide Mistral-7B with clear and actionable guidance, while it was less effective with `plan-and-solve`, `contextual-explanation` and `bullet-points`.

The above findings show that the SLMs tend to prefer different styles, e.g., bullet pointed format works better with GPT4o-mini while less well with Phi3-mini and Mistral-7B. Therefore, it is important to match the prompt styles with the preferred format of the SLMs. This observation may extend beyond just blueprints and provide insights to general prompting techniques of SLMs. For the list of detailed descriptions of all blueprint styles and the corresponding prompts, please refer to Appendix A.1.

5. Conclusion

We presented a blueprint generation and prompt template search framework designed to improve the reasoning capabilities of SLMs and addressing their sensitivity to prompt variations. By providing structured guidance, our approach has shown consistent improvements to SLM capabilities across multiple domains (math, coding, reasoning). Our approach effectively improves the reasoning capabilities of SLMs, offering a lightweight solution for on-device or resource-constrained environments. Future works include extending to more domains and SLMs, with the goal of enhancing SLMs on a wide range of applications.

Impact Statement

This work contributes to the growing effort to make generative AI more efficient, accessible, and sustainable by enhancing the reasoning capabilities of SLMs without re-

quiring additional training or model size increases. Our lightweight, training-free framework aligns closely with the mission by enabling more effective use of SLMs in resource-constrained settings. By demonstrating how structured prompting and template selection can significantly boost performance, we help advance the broader goal of achieving competitive, efficient, and privacy-aware alternatives to LLMs. We believe this approach supports responsible, scalable deployment of AI across diverse applications.

References

Abdin, M., Jacobs, S. A., Awan, A. A., Aneja, J., Awadallah, A., Awadalla, H., Bach, N., Bahree, A., Bakhtiari, A., Behl, H., et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

Agarwal, E., Dani, V., Ganu, T., and Nambi, A. Promptwizard: Task-aware agent-driven prompt optimization framework. *arXiv preprint arXiv:2405.18369*, 2024.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Brown, T. B. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Deng, M., Wang, J., Hsieh, C.-P., Wang, Y., Guo, H., Shu, T., Song, M., Xing, E. P., and Hu, Z. Rlprompt: Optimizing discrete text prompts with reinforcement learning. *arXiv preprint arXiv:2205.12548*, 2022.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pp. 5547–5569. PMLR, 2022.

Fernando, C., Banarse, D., Michalewski, H., Osindero, S., and Rocktäschel, T. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*, 2023.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Lester, B., Al-Rfou, R., and Constant, N. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*, 2021.

Long, J. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*, 2023.

Ma, R., Wang, X., Zhou, X., Li, J., Du, N., Gui, T., Zhang, Q., and Huang, X. Are large language models good prompt optimizers? *arXiv preprint arXiv:2402.02101*, 2024.

Magister, L. C., Mallinson, J., Adamek, J., Malmi, E., and Severyn, A. Teaching small language models to reason. *arXiv preprint arXiv:2212.08410*, 2022.

OpenAI. Hello gpt-4o. https://openai.com/index/hello-gpt-4o/, 2024a.

OpenAI. Gpt-4o mini: Advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence 2024b.

Pryzant, R., Iter, D., Li, J., Lee, Y. T., Zhu, C., and Zeng, M. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*, 2023.

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

Wan, Z., Wang, X., Liu, C., Alam, S., Zheng, Y., Qu, Z., Yan, S., Zhu, Y., Zhang, Q., Chowdhury, M., et al. Efficient large language models: A survey. *arXiv preprint arXiv:2312.03863*, 1, 2023.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Wei, J., Wei, J., Tay, Y., Tran, D., Webson, A., Lu, Y., Chen, X., Liu, H., Huang, D., Zhou, D., et al. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*, 2023.

Yang, S., Zhao, H., Zhu, S., Zhou, G., Xu, H., Jia, Y., and Zan, H. Zhongjing: Enhancing the chinese medical capabilities of large language model through expert feedback and real-world multi-turn dialogue. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19368–19376, 2024.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., and Ba, J. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*, 2022.

A. Appendix: Blueprint Generation and Optimization

A.1. Blueprint Generation Styles

Fig. 6 shows the blueprint generation prompt, accompanied by the 12 blueprint generation styles and their corresponding instruction prompts. The styles are chosen to reflect a diverse selection of different reasoning styles and formats. For example, the blueprints generated with styles `concrete-example` and `abstract-example` will use concrete examples or synthesized examples to illustrate detailed reasoning steps for solving a problem according to the general pattern of problem-solving for the task category. The styles `detailed-pattern`, `plain-pattern`, and `concise-highlevel` present the SLM with the instruction of pattern of steps of reasoning, in decreasing level of details, moreover, `bullet-points` will present the guidance in detailed steps and in bullet-pointed format. `instruction-focus` and `contextual-explanation` will focus on clarity by providing clarified and unambiguous instructions for the SLM to understand the problem context and reasoning steps. The `reflective-refinement`, `decision-making` and `plan-and-solve` blueprints aim to guide an SLM to perform multi-step reasoning in a single turn, such as reflection and refinement, or planning steps before formulating the solution.

In our experiments, the blueprint style selection is performed with 10 training examples for each of the 12 blueprint styles. Therefore, the number of SLM calls performed during blueprint style selection is $10 \times 12 = 120$ SLM calls.

A.2. More details on Blueprint APO Refinement

In this section, we include more details on the blueprint APO refinement procedure, as well as the detailed prompts used by $\mathcal{M}_{llm}$ for generating and refining the blueprint.

For a given task category $\mathcal{T}$ and SLM $\mathcal{M}_{slm}$, each APO round starts with an initial blueprint $\mathcal{B}$ (or a set of candidate blueprints) and works as follows:

1. **Initial blueprint evaluation:** Evaluate $\mathcal{M}_{slm}$ using the initial blueprint on sampled training data (in our experiments, we used 25 training examples).
2. **Compile error message:** Examples that $\mathcal{M}_{slm}$ answered incorrectly are compiled as an error message e with the questions, $\mathcal{M}_{slm}$ responses and correct solutions.
3. **Textual gradient:** Prompt $\mathcal{M}_{llm}$ with the task description, blueprint $\mathcal{B}$, and error message e to generate a list of *textual gradients* $\{g_j\}_{j=0}^{N_{grad}}$, i.e., each textual gradient is an error analysis of how the blueprint may have caused the error examples.
4. **Edit:** For each gradient g_j, prompt $\mathcal{M}_{llm}$ with the initial blueprint $\mathcal{B}$, the error message e and gradient g_j to generate a refined blueprint $\mathcal{B}_j^{edit}$.
5. **Paraphrase:** Prompt $\mathcal{M}_{llm}$ to write a paraphrased blueprint $\mathcal{B}_j^{para}$ for each refined blueprint $\mathcal{B}_j^{edit}$.
6. **Select:** Evaluate $\mathcal{M}_{slm}$ with the initial blueprint and all refined and paraphrased blueprints on sampled training data, selecting the best performing candidate(s).

The APO round can be repeated for multiple iterations with beam search, where the expansion steps (Step 1 to 5) expand from the initial blueprint candidates, followed by the selection step (Step 6) narrowing down to N_{beams} blueprint candidates for the next round.

In our experiments, we perform 1 round of APO, starting with the blueprint with the best style from the blueprint generation step. For Step 1 (initial blueprint evaluation), we perform evaluation of the $\mathcal{M}_{slm}$ on 25 training examples, and sample a maximum of 5 error examples for compiling the error message in Step 2. For Step 3, we generate 2 textual gradients, thereby obtaining 2 refined blueprints in Step 4, and 2 new paraphrased blueprints. In Step 6 of selection, we perform selection of the one top scoring blueprint. Due to the relatively small candidate size, the selection is performed with 20 training examples per blueprint candidate. The total number of SLM calls in the APO process is then $25 + 20 \times 5 = 125$ SLM calls. And the total number of LLM calls in the APO process is 6, with 2 calls per Step 3-5.

This optimization procedure can help us to obtain a blueprint that is refined according to the behaviors of $\mathcal{M}_{slm}$ on task $\mathcal{T}$. All prompts used for generating the textual gradient, editing and updating the blueprint, as well as paraphrasing the blueprint are included in Fig. 8. For more details on APO please refer to (Pryzant et al., 2023).

B. More Details on Template Search

To perform prompt template search, we define the search space as shown in Fig. 1, with 4 template parameters:

- Number of In-context examples = [0,1,2,3]
- Placing Task Description before In-context Examples = [True, False]
- Include Blueprint = [True, False]
- Include CoT = [True, False]

Specifically, the in-context examples are provided by the respective datasets, which includes step-by-step solutions. The inclusion of CoT (chain-of-thought) means the inclusion of the sentence *Let's think step-by-step*. In total the prompt template parameters produce 32 combinations of different prompt templates. For example, if we choose 3 in-context examples, place task description before in-context examples, include the blueprint and not include CoT, then the prompt template produced is the following: `<task-description><3-in-context-examples> <blueprint><question>`. In our experiments, we perform the successive halving template search with a reduction factor of $f = 2$, (i.e., we evaluate 32 templates in the initial iteration, then successively reduce by half in subsequent iterations until we obtain the top performing candidate). At each iteration, we sample 5 training examples for evaluating each of remaining the prompt template candidates, by applying each prompt template to form the SLM prompt. In total, the number of SLM calls incurred during prompt template search per task category per SLM model is 310 SLM calls.

C. Detailed Per-category Results on BBH

Fig. 9 shows a detailed comparison of the different method variants on each sub-category of the BBH dataset. Each plot corresponds to a task category, which shows the performance of the three SLM models (GPT4o-mini, Mistral-7B, Phi3-mini) using different method variants.

Overall we can observe a trend where the mean accuracy increase from left to right (the leftmost two bars corresponds to the CoT (1-shot) and CoT (3-shot) baselines, while the other three bars correspond to our methods, BP w.o. APO, BP w. APO and BP w. APO + Template Search, respectively). Comparing our method with CoT (3-shot), we can observe significant task improvement in several categories. For example, on Dyck Languages (Fig. 9e), GPT4o-mini shows 25% performance improvement with BP (w. APO) and a further 4% improvement with template search. On Geometric Shapes (Fig. 9g), Mistral-7B shows a 14% with BP (w. APO) and 10.5% improvement with BP (w. APO) + Template Search. On Logical Deduction Three Objects (Fig. 9k), Phi3-mini shows a 12.5% improvement with BP (w. APO) and a further 2% improvement with template search. Sometimes template search may not find the optimal template with the highest average accuracy across all variants, as a result of limited number of training data for evaluating the templates. Nevertheless, the template search procedure can usually identify a template with near-optimal performance as well as preventing low performing templates. Therefore we observe that it displays the most robust and overall best performance across all task categories and SLMs.

D. More Probing Experiment Results on Prompt Sensitivity

Apart from observing that the changing the ordering of task description and in-context examples can significantly change task performance, we also present the effect of varying the number of in-context examples on the task performance on a subset of BBH datasets. In Figure 7, we observe that increasing the number of in-context examples from 0 to 3 often results in improved performance, for example, on MBPP both GPT4o-mini and Mistral-7B shows improved performance while Phi3-mini's performance is relatively stable. For Sports Understanding (BBH), all SLMs show significantly improved performance with increasing number of examples. Sometimes increasing number of examples may results in lower performance, for example, in Temporal Sequence (BBH), Phi3-mini shows increasing performance as we increase the number of examples from 0 to 1, while we see decreasing performance as we increase the number of examples from 1 to 3, whereas Mistral-7B displays opposite trends.

E. More Details on Experiment Setup

The sizes of the SLMs are: Phi3-mini has 3.8B parameters, Mistral-7B has 7B parameters, we do not know the size of the GPT4o-mini model. For each SLM model, we use the same hyper-parameter of temperature=0 and top-p=1. For a fair comparison, we use the same hyper-parameters across all the APO variants (including our variants that optimize the

blueprint and the baseline variant that optimizes the task description), details on the APO steps, hyper-parameters, and number of LLM/SLM calls are presented in Section A.2. Details on template search are presented in Section B. For each dataset within GSM8K, MBPP, and each sub-category of BBH, we use 50 training examples, and evaluate all variants on the same held-out test set. For efficient evaluation on a wide task diversity, we use 200-300 evaluation datapoints across a total of 28 task categories and report the average accuracy across all datapoints for each category (except for BBH which we report the average over all sub-categories in Table 1), therefore, the total datapoints evaluated for each SLM and method variant is around 5600). In detail, for GSM8K, we used the same 300 evaluation datapoints from Huggingface for evaluating all SLM/method combinations. For MBPP, we used the sanitized release of MBPP dataset from Huggingface, including 257 evaluation datapoints. For BBH, we use the dataset from Huggingface, with 50 training datapoints, and 200 evaluation datapoints per sub-category (for two sub-categories with fewer than 200 evaluation datapoints, we use all available datapoints). We used regular expressions for efficient evaluation on multiple-choice and calculation problems (26 out of 27 BBH sub-categories), where the expected answers are well-defined. The word sorting category was excluded due to the variability in natural language responses that could not be evaluated reliably using regular expressions.

Blueprint Generation Prompt:
You are helping a small language model to solve some tasks: **<task_description_placeholder>**. The following are some examples:\n**<examples_placeholder>**\n**<style_instruction_placeholder>**
Wrap your generated blueprint with tokens <START> and <END>.

Blueprint styles and corresponding instructions

- **concrete_example:** Provide a blueprint using one or a few specific example questions, detailing each step of the solution process with clear, explicit instructions and thorough explanations for how to carry out each step.
- **abstract_example:** Generate a blueprint using an abstract example, detailing each step of the solution process with clear, explicit instructions and thorough explanations for how to carry out each step.
- **detailed_pattern:** Create a detailed blueprint that outlines a common pattern of steps used to solve the similar problems of this type, with explanations and rationale for each step."
- **plain_pattern:** Please generate a blueprint (without markdown) within 300 words which shows a general pattern of steps for solving tasks like these.
- **concise_highlevel:** Develop a concise, high-level guide for solving this particular type of problem, focusing on the key steps without detailed solutions.
- **instruction_focus:** Generate a blueprint with a strong focus on instructional clarity, breaking down the task into distinct instructions that are easy to follow and minimize ambiguity.
- **contextual_explanation:** Generate a blueprint that guides the small language model to clarify the contexts for the problem, providing a detailed explanation to help the small language model to understand the problem better.
- **reflective_refinement:** Develop a blueprint that guides the small language model to first attempt a solution and then critically reflect on each step to identify potential errors or areas of improvement. The model should assess the correctness of the solution then propose refined solution based on the reflection.
- **workflow:** Generate a blueprint that provides a structured workflow for solving this type of problems, detailing each step in a sequence that logically progresses from the beginning to the end, outlining the process in a systematic and organized manner.
- **bullet_points:** Provide a blueprint using a bullet point format to list the key considerations, steps, and important details to keep in mind when solving a problem.
- **decision_making:** Generate a blueprint that guides the small language model on how to systematically break down a similar task into smaller components, reason through each part, and how to make decisions at each step based on analysis and criteria.
- **plan_and_solve:** Generate a blueprint that guides the small language model to first clearly define the goal of the problem, break down the information provided, and clarify the goal and decision criteria. Once this plan is established, the model should use it to guide a step-by-step solution, ensuring each step aligns with the defined goal and criteria.

Figure 6. Blueprint generation prompt, blueprint styles and their corresponding generating instructions. When generating a blueprint, take a blueprint style and the corresponding generating instruction, which replaces the <style-instruction-placeholder> to form part of the blueprint generation prompt. The prompt is then used as an input to the LLM for generating the blueprint.
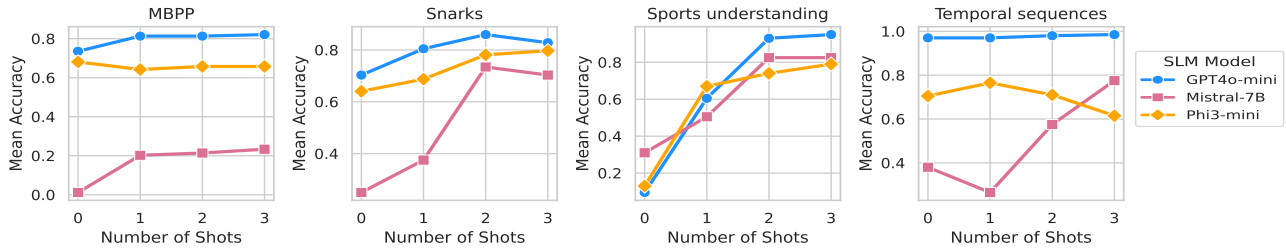
Figure 7. Sensitivity of SLMs to prompt variations across tasks - the number of in-context examples can significantly impact the SLM performance across different tasks. For instance in the temporal sequences dataset (BBH), GPT4o-mini shows relatively stable a d slightly improved performance with increasing number of examples, Mistral-7Bperformance dropped from 0-shot to 1-shot, while the performance improved from 1-shot to 3-shot, whereas Phi3-miniperformed best with 1-shot example and its performance dropped from 1-shot to 3-shot examples. In contrast, on Sports understanding dataset (BBH), all models show improved performance with increasing number of in-context examples.

Gradient Prompt: (blueprint error analysis based on SLM error)
Here is a blueprint in a prompt for a small language model (SLM) to solve a set of tasks on
<task_description_placeholder>:**<blueprint_placeholder>**.
But the prompt gets the following examples wrong.**<error_msg_placeholder>**.
Identify **<num_of_gradients>** specific reasons why the prompt is incorrect or ineffective. For each reason, consider: 1. pinpoint the error: Describe exactly what part of the blueprint or approach is causing the mistake. Mention if there is a misunderstanding, a missing step, or an incorrect interpretation of the task. 2. provide actionable suggestions: For each identified reason, suggest a specific change or addition to the blueprint that could help correct the error. Wrap each reason with <START> and <END>.

Edit Prompt: (blueprint update based on error analysis)
Here is a blueprint in a prompt for a small language model (SLM) to solve a set of tasks on
<task_description_placeholder>:**<blueprint_placeholder>**.
However the SLM gets the following examples wrong.**<error_msg_placeholder>**.
Here is one reason why the prompt is wrong and how it can be improved:
<gradient_placeholder>.
Generate an improved blueprint for the SLM to solve the tasks in general, addressing the issues mentioned in the gradient. Ensure the new blueprint includes clear, specific instructions or steps that rectify the identified errors. Focus on improving clarity and precision rather than paraphrasing. Wrap your generated blueprint with tokens <START> and <END>.

Paraphrase Prompt: (simple blueprint paraphrasing)
Here is a blueprint in a prompt for a small language model (SLM) to solve a set of tasks on
<task_description_placeholder>:**<blueprint_placeholder>**. Rephrase the blueprint to make it more concise and clear, while retaining all critical steps and details necessary for the SLM to perform the tasks effectively. The rephrased blueprint should use straightforward language to ensure clarity and ease of understanding for the SLM. Avoid losing any important information or instructions that are essential for solving the tasks. Wrap your generated blueprint with tokens <START> and <END>.

Figure 8. Prompts used for blueprint refinement via Automatic Prompt Optimization (APO). During each APO round, the LLM first performs error analysis and generates a *textual gradient* using the Gradient Prompt, then refines and updates the blueprint according to the textual gradient with the Edit Prompt. Finally, the LLM paraphrase the blueprint with the Paraphrase Prompt. During execution, the placeholders are replaced with their actual values.
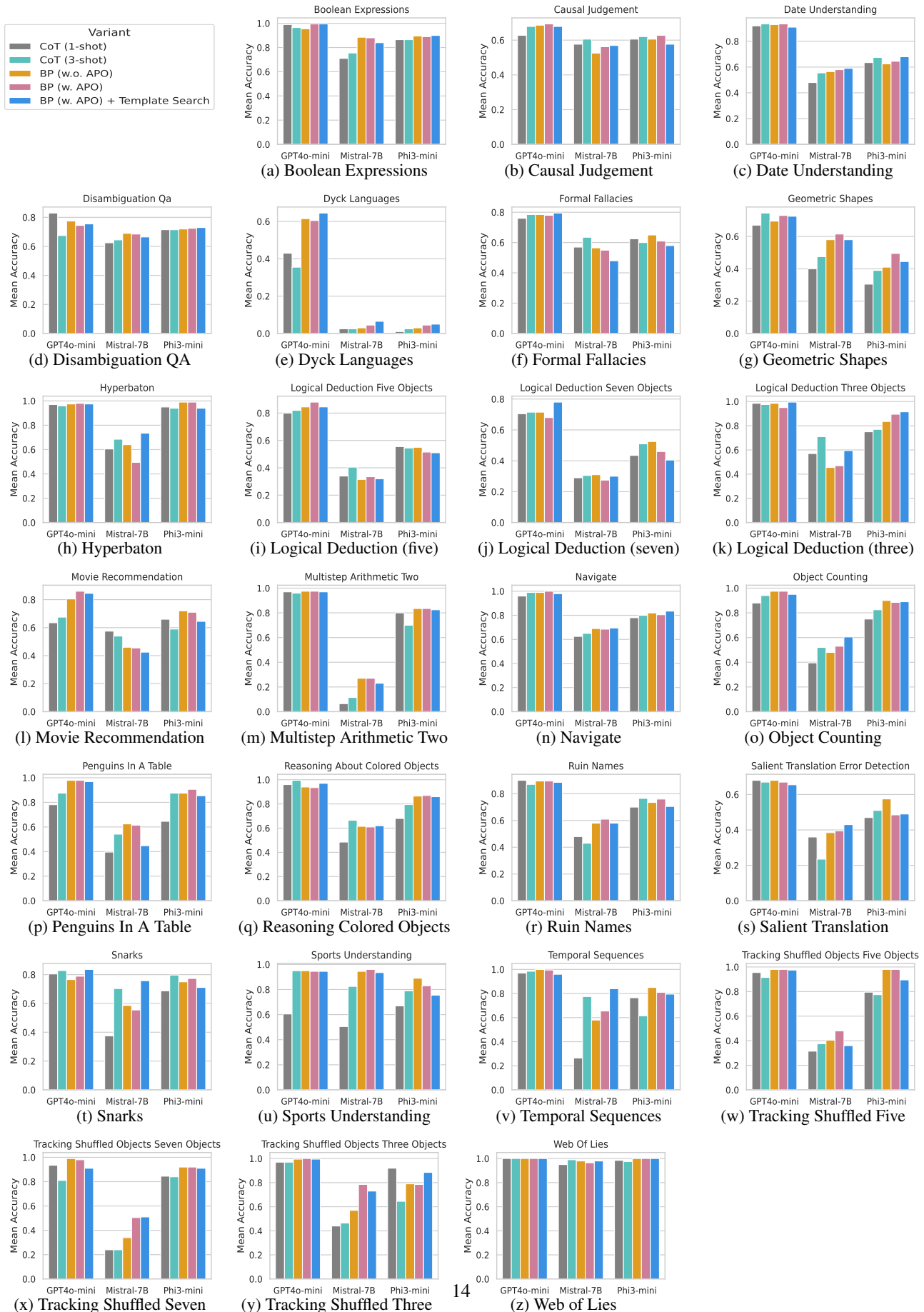
Figure 9. Per-category Performance on the BBH dataset