

# MATRYOSHKA: LEARNING TO DRIVE BLACK-BOX LLMs WITH LLMs

**Anonymous authors**

Paper under double-blind review

## ABSTRACT

Despite the impressive generative abilities of black-box large language models (LLMs), their inherent opacity hinders further advancements in capabilities such as reasoning, planning, and personalization. Existing works aim to enhance LLM capabilities via domain-specific adaptation or in-context learning, which require additional training on accessible model parameters, an infeasible option for black-box LLMs. To address this challenge, we introduce `Matryoshka`, a lightweight white-box LLM controller that guides a large-scale black-box LLM generator by decomposing complex tasks into a series of intermediate outputs. Specifically, we consider the black-box LLM as an environment, with `Matryoshka` serving as a policy to provide intermediate guidance through prompts for driving the black-box LLM. `Matryoshka` is trained to pivot the outputs of the black-box LLM aligning with preferences during iterative interaction, which enables controllable multi-turn generation and self-improvement in optimizing intermediate guidance. Empirical evaluations on three diverse tasks demonstrate that `Matryoshka` effectively enhances the capabilities of black-box LLMs in complex, long-horizon tasks, including reasoning, planning, and personalization. By leveraging this pioneering controller-generator framework to mitigate dependence on model parameters, `Matryoshka` provides a transparent and practical solution for improving black-box LLMs through controllable multi-turn generation using white-box LLMs.

## 1 INTRODUCTION

Most of the commercial large language models (LLMs) (Radford et al., 2019; Brown, 2020; Achiam et al., 2023; Chowdhery et al., 2023; Team et al., 2023; Reid et al., 2024) are *black-box* models (Sun et al., 2024b; Zhuang et al., 2024), where the model structure, parameters, or even output logits are not accessible. Although these black-box LLMs have exhibited remarkable efficacy across a diverse array of applications, revolutionizing natural language processing tasks such as text completion (Radford et al., 2019; Brown, 2020), translation (Zhu et al., 2023), question-answering (Hendrycks et al., 2020), etc, the applications of black-box LLMs continue to face significant challenges when faced with tasks that require more advanced cognitive capabilities, particularly in the realms of reasoning (Hendrycks et al., 2021; Wang et al., 2024b), planning (Valmeekam et al., 2022; Zhuang et al., 2023; Jimenez et al., 2023; Mialon et al., 2023), and personalization problems (Salemi et al., 2023; Tan et al., 2024a). Enhancing such capabilities within black-box LLMs presents unique challenges, primarily due to the *lack of direct access to internal model parameters* (Huang et al., 2023; Sun et al., 2024b; Zhuang et al., 2024). This opacity introduces substantial complexity in efforts to refine and augment these advanced cognitive functions within the framework of black-box architectures.

Existing research efforts for improving black-box LLM performance can be largely categorized into two main methodological paradigms (Figure 1): (1) **In-context learning (ICL)-based methods** (Sun et al., 2024a; Tan et al., 2024b; Zhuang et al., 2024) that are designed to guide LLM in exhibiting specific capabilities or adhering to particular directives. However, these frameworks necessitate *meticulously constructing few-shot demonstrations or prompts* for LLMs to emulate or follow, rather than fundamentally advancing their intrinsic capabilities. (2) **Adapter-based methods** (Sun et al., 2024b; Zhuang et al., 2024; Shi et al., 2024) that exploit the inherent randomness in LLM generation, producing multiple candidate outputs and subsequently selecting those that optimally satisfy domain-predetermined criteria. Nevertheless, these approaches are highly dependent on the intrinsic synthetic capabilities or built-in functionalities of the black-box LLM, potentially resulting in the selection of a

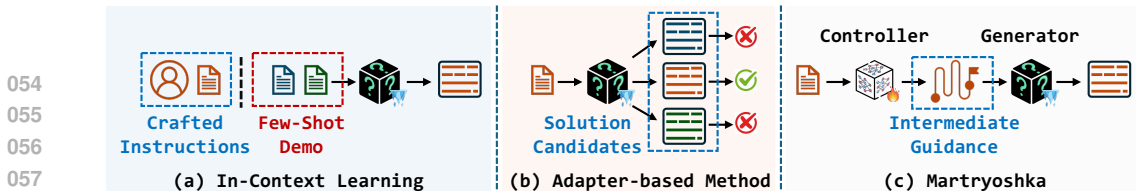


Figure 1: Enhancement in black-box LLMs capabilities. Existing methods either (a) integrate well-crafted instructions or meticulously-picked few-shot demonstrations as guidance or (b) exploit randomness in model generations to identify the most promising solution from candidates. In Matryoshka, we present (c) a controller-generator framework that enables white-box LLMs to drive the behavior of black-box LLMs for enhanced capabilities. 🔥 indicates the trainable parameters, whereas ❓ indicates the inaccessible fixed parameters.

*suboptimal candidate* when all the generated options are less than ideal. Furthermore, both ICL and adapter-based methodologies exhibit significant limitations when applied to long-horizon tasks (*e.g.*, multi-step reasoning, long-term planning, *etc.*) due to their inherent lack of environmental interaction capabilities. In light of these constraints, we propose to leverage smaller, open-source LLMs as controllers to generate soft prompts as guidance, instead of relying on hard memory in context.

Similar to the scratchpad in `o1-preview` (OpenAI, 2024c)<sup>1</sup>, we propose Matryoshka, a modular framework designed to enhance the advanced problem-solving capabilities of black-box LLMs via controllable multi-turn generations. Matryoshka consists of a lightweight white-box LLM that functions as a **controller** and a black-box LLM that serves as a **generator** or **solver**. Upon receiving the question description as input, the controller generates intermediate outputs that augment the capabilities of the subsequent black-box LLMs. For example, the controller can decompose the original complex task into high-level subtasks in reasoning or planning scenarios, or summarize profiles from historical records for personalization tasks. By conceptualizing the following **black-box LLM as the environment**, Matryoshka generates intermediate guidance alongside the original input to derive the final result through multi-turn interactions with the environment. The feedback for the outputs from the environments distinguishes positive and negative examples of intermediate generations, which can be used for preference optimization. Notably, this optimization process is inherently self-improving through iterative sampling from prior inferences and by considering the policies from earlier iterations as reference policies. Matryoshka continually enhances the advanced capabilities of the black-box LLM through controllable multi-turn generations that iteratively interact with environmental feedback.

Extensive experiments conducted on three complex tasks demonstrate the effectiveness and generalizability of Matryoshka in improving the advanced problem-solving capabilities of black-box LLMs, with an average improvement of 3.19% in accuracy for reasoning, 7.46% in success rate for planning, and 5.82% in accuracy for personalization. Importantly, Matryoshka not only enhances the capabilities of black-box LLMs without requiring access to model parameters, but also facilitates online feedback with environmental interactions. We summarize the main contributions as follows:

- **i)**, We introduce Matryoshka, one of the first modular frameworks that employ a lightweight white-box LLM to drive the generation of a large-scale black-box LLM for complex problem-solving;
- **ii)**, Matryoshka intuitively formulates the white-box LLM as a controller and the black-box LLM as a component of the environment, facilitating long-horizon controllable generation with environmental feedback; and
- **iii)**, Matryoshka adopts on-policy learning to iteratively enhance training data quality, inherently self-improving intermediate guidance for the continual enhancement of black-box LLM capabilities.

## 2 PROBLEM FORMULATION

Our objective is to enhance the capability of a black-box LLM in solving complex, long-horizon problems by calibrating its output generation to better align with specific tasks. To achieve this, we conceptualize both the original outputs and the optimal solutions as distributions within a joint space,  $\mathcal{Y} \sim \mathcal{Y}^{\text{org}} \times \mathcal{Y}^{\text{sol}}$ , where  $\mathcal{Y}^{\text{org}}$  and  $\mathcal{Y}^{\text{sol}}$  represent the original text generations and target solutions,

<sup>1</sup>Scratchpad is a sequence of intermediate chain-of-thoughts generated prior to producing the final answer. In Matryoshka, we broaden the definition of intermediate tokens to encompass various forms of guidance that can enhance the capabilities of LLMs, including task decomposition and user history summarization.

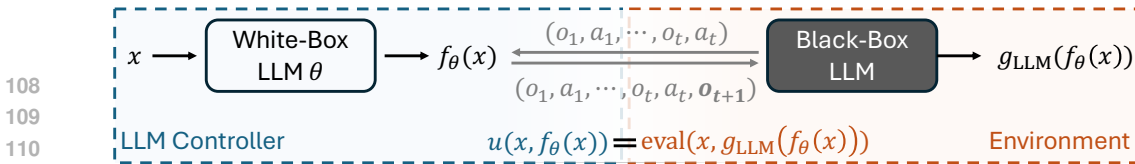


Figure 2: Controller-generator framework in Matryoshka comprising a white-box LLM as the controller and a black-box LLM as the generator and part of the environment. Given an input query  $x$ , Matryoshka leverages the intermediate generation  $f_\theta(x)$  from the controller  $\theta$  to drive the generator’s behavior. The final answer is derived from the generation  $y \sim g(f_\theta(x))$ .

respectively. Specifically, given a set of task descriptions  $\mathcal{D} = \{x_i\}_{i=1}^N$ , our goal is to adjust the outputs  $\hat{y}_i \in \mathcal{Y}^{\text{org}}$  of the black-box LLM toward the hidden target solutions  $y_i \in \mathcal{Y}^{\text{sol}}$  that successfully solve the problems. This involves driving the black-box LLM to generate outputs more closely aligned with the desired solutions without requiring access to its internal parameters.

**White-Box LLM as Controller/Policy.** We propose utilizing a lightweight white-box language model as a controller to enhance the capabilities of black-box LLMs in solving various tasks. The process begins by feeding a text-grounded task description  $x$  from the task space  $\mathcal{X}$  into a smaller language model  $\theta$ , which acts as the controller. This smaller model generates  $f_\theta(x)$ , an automatic prompt designed to augment the performance of black-box LLMs on the specific task. These prompts can facilitate various functions, such as chain-of-thoughts for reasoning, task decomposition for planning, and user profile summarization from historical records for personalization. The generated intermediate prompt  $f_\theta(x)$  is then combined with the original problem description  $x$  and input  $(x, f_\theta(x))$  into the black-box LLM. The capability enhancement will be measured by evaluating the performance improvements achieved through this controller-generator framework.

We emphasize that unlike works focusing on token-level LLM policy (Wang et al., 2024a; Rafailov et al., 2024a), our action space consists of entire intermediate guidance generations to solve the task.

**Black-Box LLM as Environment.** We recognize the black-box LLM as an environment to be controlled by the white-box LLM policy. After inputting the prompts  $(x, f_\theta(x))$ , the black-box LLM produces a final solution  $\hat{y} = g_{\text{LLM}}(x, f_\theta(x))$  for the task. We utilize the final correctness of the black-box LLM’s output to evaluate the quality  $u(x, f_\theta(x))$  as the reward of the intermediate guidance produced by the white-box LLM controller (Figure 2):

$$u(x, f_\theta(x)) := \text{eval}(x, g_{\text{LLM}}(x, f_\theta(x))), \quad (1)$$

where  $\text{eval}(\cdot)$  denotes the oracle evaluation function of the final answer. For example, in question-answering tasks with ground-truth final answer  $y$ , the evaluation function measures accuracy by comparing the prediction with ground truth as  $\text{eval}(x, g_{\text{LLM}}(x, f_\theta(x))) = \mathbb{1}(g_{\text{LLM}}(x, f_\theta(x)) = y)$ , where  $\mathbb{1}(\cdot)$  is the indicator function. For planning tasks without a ground-truth solution, the evaluation function assesses the success rate after executing the final solution as  $\text{eval}(x, g_{\text{LLM}}(x, f_\theta(x))) = \mathbb{1}_{\text{succ}}(g_{\text{LLM}}(x, f_\theta(x)))$ .

**Multi-Turn Interaction.** The above interaction between the white-box LLM controller and the black-box environment can be repeated for multi-turns for long-horizon tasks.

For initialization, a prompt  $x$  is sampled from task space  $\mathcal{X}$  and serves as the initial state  $s_0 = x$ . At each subsequent step  $t \in [T]$ , the controller generates prompts  $a_t$  based on the current  $s_{t-1}$ . In response to the controller’s action, the environment returns an observation  $o_t$  based on the history  $s_{t-1}$  and the current action  $a_t$ . The state then transitions to include the new action and observation:

$$s_t = (s_{t-1}, a_t, o_t) = (x, a_1, o_1, s_1, \dots, a_t, o_t), \quad (2)$$

and the next step begins. This process repeats for  $T$  rounds, resulting in a trajectory:

$$\tau = (x, a_1, o_1, s_1, \dots, o_T, s_T), \quad (3)$$

and we obtain the reward for the whole trajectories, according to some  $\text{eval}(\cdot)$ .

The framework formulates a Markov Decision Process (MDP), which offers the potential for solving tasks that require long-horizon generations, including long-term planning and multi-step reasoning. By obtaining feedback from  $\text{eval}(\cdot)$ , we can conduct multi-turn optimization over the white-box LLM controller on the intermediate generations. Additionally, the multi-turn interaction with the environment during the data sampling stage can help improve data quality. Although optimizing this guidance presents challenges due to the inaccessibility of the black-box LLM’s parameters that preclude backpropagation of gradients during training, the existing reinforcement learning techniques, e.g., Schulman et al. (2017); Rosset et al. (2024), can be used for policy optimization.

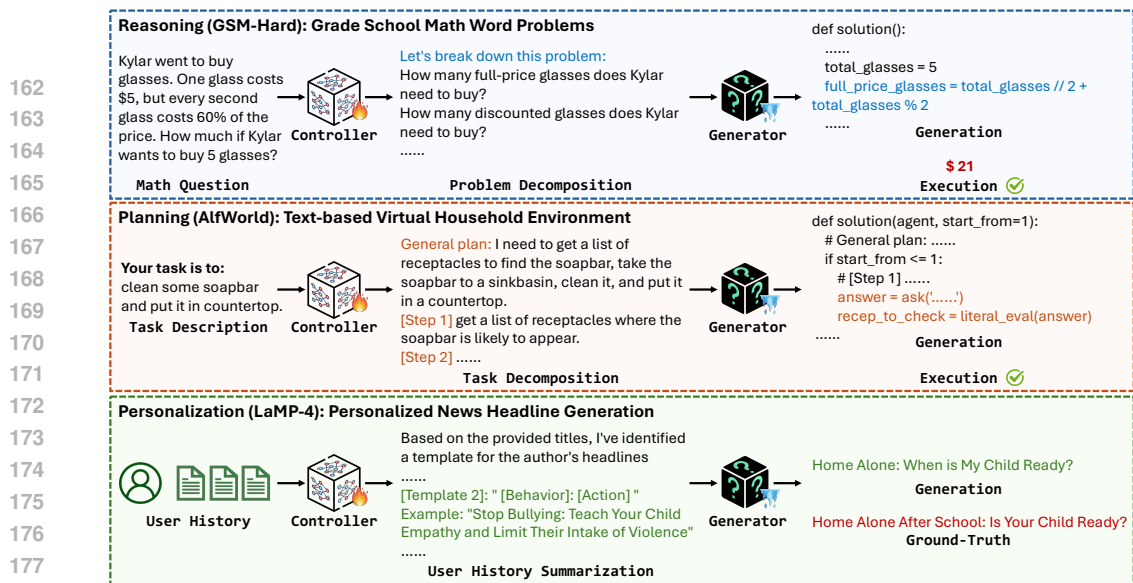


Figure 3: Examples of intermediate guidance generated by Matryoshka for complex reasoning, planning, and personalization tasks.

### 3 MATRYOSHKA

In this section, we specialize the white-box LLM controller that generates intermediate guidance to assist in task understanding and problem-solving in Section 3.1 and discuss the data collection procedure by interacting with black-box LLM in Section 3.2, which will be used for Matryoshka training to align the outputs of the black-box LLM with preferences in Section 3.3.

#### 3.1 INSTANTIATION OF WHITE-BOX LLM CONTROLLER

We instantiate the white-box LLM as a controller to generate additional guidance that assists the black-box LLM in understanding and solving a diverse range of problems. Given the varying complexity and distinct characteristics of different tasks, the controller should be capable of generating guidance in various formats. We provide examples corresponding to reasoning, planning, and personalization tasks (Figure 3):

**Problem Decomposition for Reasoning.** For reasoning tasks, generating a sequence of reasoning steps is essential to solve the problem effectively. Existing works (Zhou et al., 2023) have observed that models often perform poorly on tasks that require solving problems more complex than the exemplars provided in the prompts. To enable the model to develop better reasoning and overcome the easy-to-hard generalization issue, one strategy is to decompose complex problems into a series of simpler sub-problems and solve them sequentially. Therefore, for reasoning tasks, the white-box LLM controller outputs decomposed sub-tasks to assist the subsequent black-box LLM generator in enhancing its reasoning capabilities.

**High-Level Plan for Planning.** For planning tasks, LLMs are required to generate a sequence of actions that constitute a plan to solve the given problems. A common strategy (Sun et al., 2024a; Zhao et al., 2024) is to apply hierarchical planning for complex solutions, where a high-level planner decomposes the task into sub-goals, and a low-level planner generates a sequence of admissible actions corresponding to each specific sub-goal. To enhance the black-box LLM’s planning capabilities, we leverage the white-box controller to generate high-level plans as guidance to simplify the problems.

**User History Summarization for Personalization.** For personalization tasks, LLMs are required to tailor outputs to individual users. Existing work (Richardson et al., 2023) accomplishes this by concatenating the user’s input query with a profile summarizing the user’s preferences and behavior patterns. To enhance the black-box LLM’s personalization capabilities, we utilize the white-box LLM controller to generate summaries of user histories. This approach enables black-box LLMs to better understand users and generate tailored content accordingly.

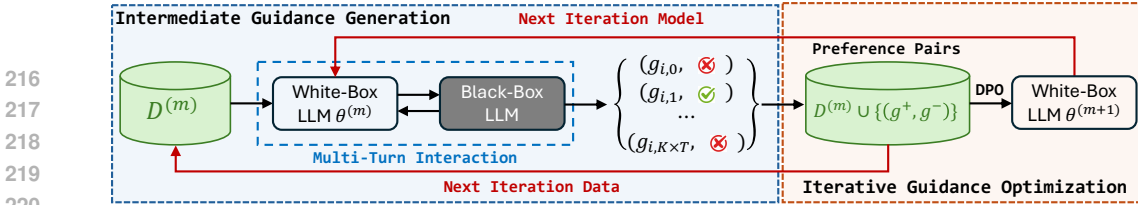


Figure 4: Overview of iterative guidance optimization. By iteratively updating both the model and the reference policy, Matryoshka progressively refines its intermediate guidance.

### 3.2 DATA COLLECTION BY INTERACTING WITH BLACK-BOX LLM ENVIRONMENT

Optimizing the intermediate guidance generated by the controller presents significant challenges for two main reasons: (1) *Lack of ground-truth guidance*: There are no ground-truth intermediate generations available to serve as supervision signals for the controller’s outputs. (2) *Uncertainty in performance improvement*: It is difficult to determine which guidance will reliably enhance the downstream performance of the black-box LLM. To address these challenges, we formulate the black-box LLM as an environment system and employ multi-turn interactions with environmental feedback during data sampling.

In the MDP formulation, we consider the *action space* as the set of possible guidance that can enhance the capabilities of black-box LLMs. The *observation space* is determined by the oracle evaluation function for each task, defined as  $\text{eval}(\cdot)$ , where the sampled supervision signal is denoted as  $z$ , with  $z = 1$  indicating that  $f_\theta(x)$  is positive guidance while  $z = 0$  indicating  $f_\theta(x)$  negative guidance. During the multi-turn interactions, if the observation  $o_t$  at the  $t$ -th step returns a negative signal, the next action step  $a_{t+1}$  involves modifying the intermediate guidance based on the feedback. The interactions continue until a positive signal is observed or the maximum number of interaction turns  $T$  is reached.

For each input  $x_i$ , we perform  $T$ -step multi-turn interactions with the black-box LLM-based environment to obtain the trajectories  $(a_{i,1}, o_{i,1}, a_{i,2}, o_{i,2}, \dots, a_{i,T}, o_{i,T})$ . To increase the diversity of intermediate generations, we introduce randomness into the policy and repeat the entire interaction process  $K$  times. This results in  $K$  trajectories, yielding intermediate generations  $\{g_{i,1}, g_{i,2}, \dots, g_{i,K \times T}\}$  along with their corresponding observations  $\{o_{i,1}, o_{i,2}, \dots, o_{i,K \times T}\}$ , which serve as sampling signals. We then sample the positive guidance  $g_i^+$  from the set of guidance with positive observations,  $g_i^+ \sim \{g_{i,j} | o_{i,j} = 1\}$  and the negative guidance from the remaining generations,  $g_i^- \sim \{g_{i,j} | o_{i,j} = 0\}$ .

### 3.3 ITERATIVE GUIDANCE OPTIMIZATION

As white-box LLMs like LLaMA are pre- and post-trained for general purposes, they may struggle to fulfill the specific tasks required by the controller. Additionally, there may be discrepancies between what the controller considers “good” guidance and what the generator interprets as “good” guidance. To this end, the guidance generated by the white-box LLM controller needs further optimization to enhance the performance of the black-box LLM generator.

**Supervised Fine-Tuning for Behavior Cloning.** To quickly initialize the controller’s policy, we adopt the concept of behavior cloning (BC) from reinforcement learning, which involves learning an initial policy by imitating the actions of an expert agent. This is typically achieved through supervised learning on a set of curated instruction-completion pairs for LLMs. We leverage the capabilities of more advanced models, such as GPT-4 (OpenAI, 2024a), to generate the desired guidance for the black-box LLMs on a small set of samples. This data is then used to perform supervised fine-tuning (SFT) on the white-box LLM controller as an initial warm-up step:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(x,g) \sim \mathcal{D}_{\text{SFT}}} \left[ \sum_{l=1}^L \log f_\theta(g_l | g_{<l}, x) \right]. \quad (4)$$

Through this SFT process, the white-box LLM controller begins to acquire the capability to effectively guide the subsequent black-box LLM. It can then be utilized to generate high-quality guidance for further optimization steps.

**Direct Guidance Optimization.** By allowing the warmed-up white-box LLM controller to interact with the black-box LLM environment over multiple turns, we can curate a dataset containing



both “good” and “bad” guidance pairs from `Matryoshka`’s intermediate generations. Following reinforcement learning with human feedback (RLHF) (Bai et al., 2022; Ouyang et al., 2022; Ziegler et al., 2019), we obtain relative feedback by comparing pairs of generated guidance that share the same prompt. The preference signal is modeled using the Bradley-Terry model (Bradley & Terry, 1952). Given an input  $x$  and a generated guidance pair  $(g^+, g^-)$ , the model specifies the probability of  $g^+$  being chosen over  $g^-$  as:

$$p(g^+ \succ g^- | x) = \frac{\exp(u(g^+; x))}{\exp(u(g^+; x)) + \exp(u(g^-; x))} = \sigma(u(g^+; x) - u(g^-; x)), \quad (5)$$

where  $\sigma(x) = \frac{e^x}{e^x + 1}$  is the logistic function. This formulation allows us to access sequence-level preferences to optimize the intermediate guidance generated by the white-box LLM controller. Following Rafailov et al. (2024b), we establish a connection between the white-box LLM controller and its associated optimal policy. Specifically, we consider the following KL-regularized planning problem with respect to a reference policy  $\pi_{\text{ref}}$ :

$$\max_{\theta} \mathbb{E}_{x \sim \mathcal{X}} \mathbb{E}_{g \sim \pi_{\theta}(\cdot | x)} [u(x, g) - \eta^{-1} \mathbb{D}_{\text{KL}}[\pi_{\theta}(g | x) || \pi_{\text{ref}}(g | x)]]. \quad (6)$$

The optimization problem above has a closed-form solution. For any guidance  $g$ , the optimal policy  $\pi^*$  is given by:

$$\pi^*(g | x) \propto \pi_{\text{ref}}(g | x) \exp(\eta u(x, g)). \quad (7)$$

This leads us to the direct preference optimization (DPO) loss for optimizing the intermediate guidance generated by the white-box LLM controller:

$$\mathcal{L}_{\text{DPO}} := \mathbb{E}_{(x, g^+, g^-) \sim \mathcal{D}} \left[ -\log \sigma \left( \eta^{-1} \left[ \log \left( \frac{\pi_{\theta}(g^+ | x)}{\pi_{\text{ref}}(g^+ | x)} \right) - \log \left( \frac{\pi_{\theta}(g^- | x)}{\pi_{\text{ref}}(g^- | x)} \right) \right] \right) \right]. \quad (8)$$

**Iterative Guidance Optimization.** In the previous sections, we discussed sampling positive and negative intermediate generations through multi-turn interactions with the environment. However, an imbalance between positive and negative samples may arise, leading to overfitting on simplistic patterns and hindering the self-improvement of the white-box LLM controller. To address this issue, we propose an iterative guidance optimization method (Figure 4) that interleaves data sampling and training steps. We begin by initializing the model with parameters  $\theta^{(0)} = \theta$  without any prior training and sample an initial dataset  $\mathcal{D}^{(0)}$  as introduced in Section 3.2. At the  $m$ -th iteration, we have the optimized model  $\theta^{(m)}$ . Following STaR (Zelikman et al., 2022), we enhance the model’s generation for the next iteration by bootstrapping the dataset. This involves combining the previous datasets with new samples generated by the current model  $\{g_{i,0}^{(m)}, g_{i,1}^{(m)}, \dots, g_{i,K \times T}^{(m)}\} \sim f_{\theta^{(m)}}(x)$ :

$$\begin{cases} \mathcal{D}_+^{(m)} = \{g_{i,j}^{(m)} | o_{i,j}^{(m)} = 1\} \cup \mathcal{D}_+^{(m-1)}, \\ \mathcal{D}_-^{(m)} = \{g_{i,j}^{(m)} | o_{i,j}^{(m)} = 0\} \cup \mathcal{D}_-^{(m-1)}, \end{cases} \quad (9)$$

In the  $t$ -th iteration, we construct the training dataset  $\mathcal{D}^{(t)}$  for DPO by sampling positive and negative pairs for each question. When training the model for the next iteration  $\theta^{(m+1)}$ , we update the reference policy to be the model from the previous iteration,  $\pi_{\theta^{(m)}}^{(m)}(g | x)$ . Consequently, the training objective for iterative guidance optimization of the white-box LLM controller becomes:

$$\mathcal{L}_{\text{IGO}} := \mathbb{E}_{(x, g^+, g^-) \sim \mathcal{D}} \left[ -\log \sigma \left( \eta^{-1} \left[ \log \left( \frac{\pi_{\theta^{(m+1)}}^{(m+1)}(g^+ | x)}{\pi_{\theta^{(m)}}^{(m)}(g^+ | x)} \right) - \log \left( \frac{\pi_{\theta^{(m+1)}}^{(m+1)}(g^- | x)}{\pi_{\theta^{(m)}}^{(m)}(g^- | x)} \right) \right] \right) \right].$$

## 4 EXPERIMENTS

In this section, we present comprehensive experiments on a diverse set of complex, long-horizon tasks to demonstrate the enhanced capabilities of black-box LLMs using `Matryoshka`.

#### 4.1 EXPERIMENTAL SETUP

**Tasks and Datasets.** We consider three types of tasks in experiments, each targeting a distinct capability of black-box LLMs: (1) *LaMP* (Salemi et al., 2023) for personalization capabilities, (2) *GSM8K* (Cobbe et al., 2021) for reasoning capabilities, and (3) *ALFWorld* (Shridhar et al., 2020) for planning capabilities. Dataset details are available in Appendix C.

**Baselines.** We consider the following baselines: (1) *Baselines in personalization*, we consider both one-stage and two-stage personalization models, including Profile-Augmented Generation (PAG) (Richardson et al., 2023) and Retrieval-Augmented Generation (RAG) (Salemi et al., 2023). (2) *Baselines in reasoning*, we include Chain-of-Thoughts (CoT) (Wei et al., 2022), Least-to-Most (Zhou et al., 2023), Program-Aided Language Models (PAL) (Gao et al., 2023), and PAL<sub>Self-Debug</sub> (Chen et al., 2023). (3) *Baselines in planning*, we mainly compare `Matryoshka` with BUTLER (Shridhar et al., 2020), ReAct (Yao et al., 2023), Reflexion (Shinn et al., 2023), and AdaPlanner (Sun et al., 2024a). Baseline details can be found in Appendix D.

**Evaluation Metrics.** For the personalization tasks, consistent with the evaluation metrics specified in *LaMP* (Salemi et al., 2023), we use accuracy (*Acc*) and F1 score (*F1*) for the classification tasks in *LaMP*-2N and *LaMP*-2M. For the ordinal multi-class classification task in *LaMP*-3, we employ mean absolute error (MAE) and root mean squared error (*RMSE*). To comprehensively evaluate the personalized text generation tasks in *LaMP*-4 and *LaMP*-5, we report ROUGE-1 (*R-1*), ROUGE-L (*R-L*), and *BLEU* scores. For the math reasoning task, we assess the models based on the *accuracy* of obtaining the final correct answer. For the planning task, consistent with previous works (Sun et al., 2024a), we evaluate performance using the *success rate* (%). The success rate is calculated as the number of successful episodes divided by the total number of episodes. In *ALFWorld*, an episode is considered a failure if the task remains unsolved after executing 50 actions, which is the maximum allowed number of actions per episode.

**Implementations.** For the white-box LLM controller, we utilize `LLaMA-3-8B-Instruct` as the backbone language model. In the black-box LLM environment, our experiments employ `gpt-4o-mini` for personalization tasks in *LaMP*, and `gpt-3.5-turbo` for reasoning and planning tasks in *GSM8K* and *ALFWorld*, respectively. All experiments with GPTs are conducted using the Microsoft Azure OpenAI service. Please refer to Appendix E for implementation details.

#### 4.2 PERSONALIZATION: LAMP

**Main Results.** Table 1 summarizes the primary experimental results on the *LaMP* dataset. Our proposed method, `Matryoshka`, consistently outperforms or matches other state-of-the-art baselines, highlighting its efficacy of advancing black-box LLMs in personalization. For classification tasks, `Matryoshka` achieves an accuracy of 0.832 on *LaMP*-2N and 0.535 on *LaMP*-2M, surpassing other baselines by a significant margin. For generation tasks, `Matryoshka` also attains over a 25% improvement in *BLEU* score on *LaMP*-4. These results demonstrate the effectiveness of `Matryoshka` in both classification and generative personalization tasks. Furthermore, `Matryoshka` has the potential to be enhanced with RAG, combining intermediate generations with the retrieved user history data to improve performance further.

**Ablation Studies.** In our ablation studies, we compare our proposed method, `Matryoshka`, with a baseline lacking Intermediate Guidance Optimization (IGO) in Table 1. Using the same black-box model (`gpt-4o-mini`), our optimized white-box controller consistently and significantly outperformed the original `LLaMA-3-8B-Instruct`. These results demonstrate the effectiveness of IGO in enhancing the white-box controller to generate more informative and higher-quality intermediate outputs, thereby guiding the black-box model toward better final answers.

**Plug-and-Play.** `Matryoshka` can seamlessly apply the optimized white-box controller, `LLaMA-3-8B-Instruct`, to other black-box models in a plug-and-play manner without additional training costs. We further utilize this well-tuned white-box controller as a plug-in to integrate with black-box models such as `gpt-3.5-turbo` and `gemini-1.5-flash`. Table 2 presents the plug-and-play results. The experimental results show that our well-tuned controller consistently outperforms other baselines. Specifically, on *LaMP*-3 and *LaMP*-4, our plug-in surpasses other baselines by a large margin, demonstrating effectiveness across both classification and generation

Dataset (→)	LaMP-1		LaMP-2N		LaMP-2M		LaMP-3		LaMP-4		
Method (↓)	Acc. ↑	F-1 ↑	Acc. ↑	F-1 ↑	Acc. ↑	F-1 ↑	MAE ↓	RMSE ↓	R-1 ↑	R-L ↑	BLEU ↑
gpt-4o-mini (OpenAI, 2024b)	0.514	0.513	0.655	0.473	0.413	0.325	0.371	0.673	0.132	0.116	0.992
RAG (k=1) (Salemi et al., 2023)	0.626	0.624	0.733	0.539	0.444	0.378	0.311	0.631	0.141	0.126	1.296
RAG (k=4) (Salemi et al., 2023)	<u>0.632</u>	<u>0.632</u>	0.792	<u>0.611</u>	<u>0.502</u>	0.430	<b>0.272</b>	<b>0.579</b>	<u>0.161</u>	<u>0.146</u>	<u>2.953</u>
PAG (Richardson et al., 2023)	0.624	0.624	0.775	0.559	0.496	<u>0.443</u>	0.316	0.645	0.143	0.130	1.968
Matryoshka	<b>0.640</b>	<b>0.640</b>	<b>0.832</b>	<b>0.614</b>	<b>0.535</b>	<b>0.475</b>	<u>0.282</u>	<u>0.588</u>	<b>0.171</b>	<b>0.157</b>	<b>4.144</b>
w/o IGO	0.611	0.611	<u>0.807</u>	0.575	0.496	0.432	0.311	0.636	0.131	0.120	1.341

Table 1: Main experimental results on the personalization task using the LaMP benchmark. For all baselines, including Matryoshka, we utilize gpt-4o-mini as the black-box LLM generator. R-1 and R-L refer to ROUGE-1 and ROUGE-L, respectively.  $k$  denotes the number of items retrieved. An upward arrow (↑) indicates that higher values are preferred, whereas a downward arrow (↓) signifies that lower values are better. The best score and second-best score for each task are emphasized in **bold** and underlined, respectively. IGO represents Intermediate Guidance Optimization. Notations are consistent across tables.

Dataset (→)	LaMP-1		LaMP-2N		LaMP-2M		LaMP-3		LaMP-4		
Method (↓)	Acc. ↑	F-1 ↑	Acc. ↑	F-1 ↑	Acc. ↑	F-1 ↑	MAE ↓	RMSE ↓	R-1 ↑	R-L ↑	BLEU ↑
Matryoshka (4o-mini)	<b>0.640</b>	<b>0.640</b>	<b>0.832</b>	<b>0.614</b>	<b>0.535</b>	<b>0.475</b>	<b>0.282</b>	<b>0.588</b>	<b>0.171</b>	<b>0.157</b>	<b>4.144</b>
gpt-3.5-turbo	0.590	0.589	0.790	0.594	0.399	0.325	0.357	0.693	0.166	0.150	3.433
PAG (gpt-3.5)	0.590	0.589	0.790	0.594	0.399	0.325	0.357	0.693	0.166	0.150	3.433
Plug-and-play (gpt-3.5)	<b>0.594</b>	<b>0.593</b>	<b>0.798</b>	<b>0.609</b>	0.469	0.412	<b>0.286</b>	<b>0.599</b>	<b>0.176</b>	<b>0.161</b>	<b>4.222</b>
w/o IGO (gpt-3.5)	0.585	0.585	0.790	0.608	<b>0.472</b>	<b>0.425</b>	0.334	0.670	0.160	0.147	3.015
gemini-1.5-flash	0.518	0.510	0.700	0.498	0.368	0.279	0.546	0.825	0.135	0.113	1.494
Plug-and-play (gemini)	<b>0.573</b>	<b>0.565</b>	<b>0.825</b>	<b>0.615</b>	0.504	<b>0.418</b>	<b>0.298</b>	<b>0.614</b>	<b>0.183</b>	<b>0.170</b>	<b>5.002</b>
w/o IGO (gemini)	0.568	0.561	0.811	0.602	<b>0.505</b>	0.411	0.365	0.715	0.164	0.150	3.439

Table 2: Plug-and-Play experimental results for gpt-3.5-turbo and gemini-1.5-flash across the LaMP benchmark. We employ Matryoshka pre-trained on gpt-4o-mini as the white-box LLM controller.

tasks. The effectiveness of Matryoshka in plug-and-play scenarios arises from the generalization capability of intermediate guidance, which can benefit different black-box LLMs.

**Effects of Profile Count.** To further investigate the effect of user profile count on the generation of intermediate outputs, we analyze performance across different numbers of profiles per user. Figure 5 presents the accuracy and ROUGE-L curves separately for LaMP-2M and LaMP-4, with the x-axis representing the total number of profiles per user (e.g., “0-20” indicates users with 0 to 20 profiles). We compared the results of our proposed method, Matryoshka, and PAG, utilizing the white-box controller Llama-3-8B-Instruct and the black-box model gpt-4o-mini. On LaMP-2M, as the profile count increases, PAG’s performance significantly deteriorates, whereas Matryoshka maintains stable performance and surpasses PAG by an increasing margin. For LaMP-4, both Matryoshka and PAG exhibit similar trends, but Matryoshka consistently outperforms PAG by a substantial and steady margin. These results demonstrate the efficacy of IGO in enhancing the summarization capabilities of the black-box controller, especially when dealing with varying and large amounts of profiles.

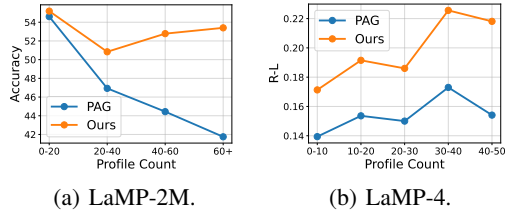


Figure 5: Effect of number of history per user in LaMP-2M and LaMP-4.

### 4.3 REASONING: GSM8K

Table 3 presents the main results on the GSM8K dataset. We employ a three-shot prompt design across all baselines, including ours. PAL<sub>Self-Debug</sub> refers to the addition of close-loop refinement to PAL during the inference stage. Our method consistently outperforms all baselines across the dataset, surpassing the strongest baseline, PAL<sub>Self-Debug</sub>, by a margin of 4.2% when using the base LLM.



This improvement stems from the optimized intermediate guidance generated by Matryoshka. Conditioned on this guidance, Matryoshka enables the black-box LLM to generate long-horizon solutions to solve the tasks. Similar to LaMP, Matryoshka trained with gpt-3.5-turbo can be seamlessly applied to other black-box models for solving mathematical problems on GSM8K without additional training costs. Notably, Matryoshka learns high-level planning abilities without focusing on specific details, which broadens its applicability.

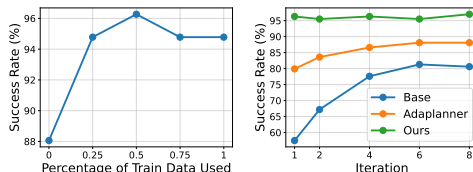
Dataset (→)	GSM8K		GSM-HARD	
	gpt-3.5	4o-mini	gpt-3.5	4o-mini
Methods (↓)				
CoT	0.809	0.932	0.406	0.500
Least-to-Most	0.811	0.908	0.425	0.498
PAL	0.802	0.920	0.638	0.748
PAL-Self-Debug	0.864	0.943	0.701	0.774
Matryoshka	<b>0.911</b>	<b>0.956</b>	<b>0.738</b>	<b>0.779</b>
w/o IGO	0.896	0.954	0.729	0.780

Table 3: Accuracy on the mathematical reasoning task using the GSM8K dataset.

#### 4.4 PLANNING: ALFWORLD

**Main Results.** Matryoshka consistently outperforms existing baselines, achieving state-of-the-art performance with an overall success rate of 95.52% on ALFWorld tasks (Table 4). This superior performance indicates that Matryoshka effectively generates plans to guide the task execution of the black-box model, enhancing its ability to interact with the environment. Furthermore, we observe that Matryoshka exhibits superior performance compared to both the untuned white-box model (w/o Guidance Optimization) and the white-box models trained with fewer rounds of Intermediate Guidance Optimization (w/o 1<sup>st</sup>/2<sup>nd</sup>-round IGO). As the number of IGO training rounds increases, Matryoshka’s performance on ALFWorld correspondingly improves, ultimately raising the success rate from 81.34% to 95.52%. These results underscore the efficacy of the IGO training employed in Matryoshka.

**Ablation Studies on Sample Efficiency.** Figure 6(a) illustrates the relationship between success rate (%) and the proportion of training data used to optimize the controller. In the ALFWorld environment, Matryoshka achieves an accuracy of 94.78% using only one-quarter of the training data, surpassing the best-performing baseline, AdaPlanner, by 6.7%. This demonstrates the sample efficiency of Matryoshka in achieving high performance with limited training data. This study demonstrates that Matryoshka significantly reduces the reliance on high-quality task planning samples and expert trajectories, thereby providing a more resource-efficient solution.



(a) Sample Efficiency. (b) Number of Turns  $M$ .

Figure 6: Success rate (%) w.r.t number of (a) training samples and (b) interaction turns.

**Effect of Number of Interaction Turns  $M$ .** We evaluate the performance of Matryoshka against the vanilla LLaMA3-8B-Instruct and AdaPlanner on ALFWorld, varying the number of closed-loop iterations  $M$  during the inference phase. As illustrated in Figure 6(b), following DPO training, Matryoshka achieves an accuracy exceeding 95% in the open-loop inference setting ( $M=1$ ), significantly surpassing both AdaPlanner and LLaMA3-8B-Instruct. Furthermore, during an 8-iteration closed-loop inference, Matryoshka maintains the highest accuracy of 97%. These findings indicate that Matryoshka is capable of generating exceptionally high-quality plans, enabling the GPT model serving as the task executor to interact with the environment and complete tasks successfully without requiring closed-loop refinement.

## 5 RELATED WORKS

**Black-Box LLMs Generation Enhancement.** Existing approaches aiming to enhance the generation capabilities of black-box LLMs can be broadly categorized into two groups: (1) *ICL*- and (2) *adapter-based* methods. ICL-based methods (Sun et al., 2024a; Tan et al., 2024a; Zhuang et al., 2024) are designed to augment the original query with carefully crafted instructions or meticulously constructed few-shot demonstrations to guide the model. While this enables the black-box LLM to exhibit specific

Methods (↓) Tasks (→)	Pick	Clean	Heat	Cool	Examine	Pick Two	All (134 tasks)
BUTLER (Shridhar et al., 2020)	46.00	39.00	74.00	<b>100.00</b>	22.00	24.00	37.00
ReAct (Yao et al., 2023)	37.50	64.52	69.57	42.86	38.89	17.65	47.76
Reflexion (Shim et al., 2023)	50.00	41.94	65.22	52.38	66.67	47.06	52.99
AdaPlanner (Sun et al., 2024a)	<b>100.00</b>	<b>93.55</b>	78.26	95.24	66.67	<b>88.24</b>	88.06
Matryoshka	<b>100.00</b>	<b>93.55</b>	<b>100.00</b>	90.48	<b>100.00</b>	<b>88.24</b>	<b>95.52</b>
w/o 2 <sup>nd</sup> -round IGO	100.00	93.55	100.00	100.00	83.33	88.24	94.78
w/o 1 <sup>st</sup> , 2 <sup>nd</sup> -round IGO	100.00	93.55	86.96	95.24	55.56	88.24	88.06
w/o Guidance Optimization	100.00	93.55	91.30	85.71	11.11	88.24	81.34

Table 4: Success rate (%) across six planning tasks from AlfWorld. For all baselines, including Matryoshka, we utilize gpt-3.5-turbo as the black-box LLM generator.

capabilities or adhere to particular directives, these methods require significant human effort in prompt engineering and result in prompts that are rigid and static. Adapter-based methods (Sun et al., 2024b; Shi et al., 2024; Zhuang et al., 2024) follow a best-of-N selection evaluation paradigm (Lightman et al., 2023). Given a problem, adapter-based methods generate  $N$  candidate solutions from the generator and subsequently evaluate them using a lightweight adapter to identify the highest-scoring solution as the final answer. However, such methods are heavily dependent on the generative capabilities of the black-box LLM, which may result in selecting a suboptimal candidate as *the best of a bad bunch*.

**Small LMs Drive LLMs Generation.** SuperICL (Xu et al., 2023) incorporates outputs from smaller language models (LMs) as complementary information for input queries, integrating them into the context provided to black-box LLMs. However, these smaller LMs are fixed and can only support classification tasks that rely on label predictions with associated confidence scores. HYDRA (Zhuang et al., 2024) is a retrieval-augmented generation framework that trains a BERT-sized reranker to reorder retrieved passages to better cater to user-specific requirements. Nevertheless, these methods apply only discrete optimization on the prompt through reranking and selection of few-shot demonstrations, which limits the potential improvements achievable via prompt engineering.

**Reinforcement Learning for Prompt Optimization.** As LLMs scale, new capabilities emerge, enabling models to learn tasks efficiently through a few in-context demonstrations. To harness these capabilities, several approaches have been proposed to leverage reinforcement learning for improved prompt generation, enhancing LLM performance. RLPrompt (Deng et al., 2022) introduces an RL-based framework for generating optimal prompts via black-box optimization. Similarly, TEMPERA (Zhang et al., 2023) formulates prompt optimization as test-time prompt editing, using RL to efficiently explore the editing space. BDPL (Diao et al., 2023) further advances this by proposing a variance-reduced policy gradient algorithm to estimate gradients of parameters in the categorical distribution of each discrete prompt. However, these methods primarily focus on classification tasks, where gradient estimation is straightforward, limiting their applicability to more complex generation tasks requiring long-horizon solutions.

## 6 CONCLUSION AND FUTURE WORK

We introduced Matryoshka, a lightweight white-box LLM controller designed to augment the capabilities of large-scale black-box LLMs across a wide range of complex tasks, including reasoning, planning, and personalization. By leveraging a controller-generator framework with environmental feedback, Matryoshka effectively decomposes complex tasks and guides black-box LLMs through intermediate guidance. Through policy gradient optimization, Matryoshka exhibits a self-improving nature that continually enhances LLM capabilities via multi-turn guidance optimization. Extensive experiments on three diverse datasets demonstrate its effectiveness in steering black-box LLMs for long-horizon tasks without requiring access to model parameters or output probabilities. Compared to the best-performing state-of-the-art baselines, Matryoshka achieves average improvements of 3.19% in reasoning tasks, 7.46% in planning tasks, and 5.82% in personalization tasks. These results underscore the potential Matryoshka as a transparent and scalable solution, enabling white-box LLMs to drive black-box LLMs in complex problem-solving. Future work could extend Matryoshka to tackle more complex applications requiring long-horizon generation and reasoning, such as solving software engineering problems and proving mathematical theorems. Additionally, the controller component of Matryoshka could be developed into a self-enhancing mechanism or a universal controller applicable to a wide range of real-world applications.

## 7 REPRODUCIBILITY STATEMENT

The datasets utilized in this study are all publicly available, including LaMP for the personalization task, GSM8K and GSM-Hard for the reasoning task, and ALFWorld for the planning task. Detailed descriptions of these datasets and their corresponding tasks are provided in Appendix C. In Appendix D, we outline the baselines used for comparison and describe the experimental setup. Appendix E offers an in-depth explanation of the main experiments, including hardware and software configurations, hyperparameter settings, and step-by-step procedures for the three tasks. Additionally, Appendix F presents case studies for each task, demonstrating the superior performance of our method compared to the baselines in a more intuitive manner. Appendix G details the prompts used for each task. The implementation of `Matryoshka` is provided in the supplementary materials and will be released publicly available on GitHub upon acceptance.

## 8 ETHICS STATEMENT

We strictly followed the data usage guidelines for interactions with Microsoft Azure’s OpenAI API and Gemini API service. Although our research relied solely on publicly available datasets, we took extra precautions to minimize any potential risk of information leakage. Specifically, we opted out of the human review process by completing and submitting the Azure OpenAI Additional Use Case Form<sup>2</sup>. This proactive measure highlights our commitment to maintaining the highest data privacy standards and ethical research practices, especially concerning personalization tasks.

## REFERENCES

- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. A general theoretical paradigm to understand learning from human preferences. In *International Conference on Artificial Intelligence and Statistics*, pp. 4447–4455. PMLR, 2024.
- Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.
- Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.
- Tom B Brown. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
- Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.
- Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models, 2024.
- Leshem Choshen, Lior Fox, Zohar Aizenbud, and Omri Abend. On the weaknesses of reinforcement learning for neural machine translation. In *International Conference on Learning Representations*, 2019.
- Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

<sup>2</sup><https://aka.ms/oai/additionalusecase>

- 594 Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser,  
595 Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve  
596 math word problems. *arXiv preprint arXiv:2110.14168*, 2021.
- 597
- 598 Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song,  
599 Eric Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement  
600 learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language*  
601 *Processing*, pp. 3369–3391, 2022.
- 602 Shizhe Diao, Zhichao Huang, Ruijia Xu, Xuechun Li, LIN Yong, Xiao Zhou, and Tong Zhang. Black-  
603 box prompt learning for pre-trained language models. *Transactions on Machine Learning Research*,  
604 2023. ISSN 2835-8856. URL <https://openreview.net/forum?id=IvsGP7xRvm>.
- 605
- 606 Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph,  
607 and Aleksander Madry. Implementation matters in deep policy gradients: A case study on ppo and  
608 trpo. *arXiv preprint arXiv:2005.12729*, 2020.
- 609 Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model  
610 alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*, 2024.
- 611
- 612 Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and  
613 Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine*  
614 *Learning*, pp. 10764–10799. PMLR, 2023.
- 615 Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek  
616 Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training  
617 (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.
- 618
- 619 Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and  
620 Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint*  
621 *arXiv:2009.03300*, 2020.
- 622 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song,  
623 and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv*  
624 *preprint arXiv:2103.03874*, 2021.
- 625
- 626 Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh  
627 Agarwal. V-star: Training verifiers for self-taught reasoners, 2024.
- 628
- 629 Yangsibo Huang, Daogao Liu, Zexuan Zhong, Weijia Shi, and Yin Tat Lee. *k* nn-adapter: Efficient  
630 domain adaptation for black-box language models. *arXiv preprint arXiv:2302.10879*, 2023.
- 631 Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik  
632 Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint*  
633 *arXiv:2310.06770*, 2023.
- 634
- 635 Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan  
636 Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. *arXiv preprint*  
637 *arXiv:2305.20050*, 2023.
- 638 Trung Quoc Luong, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. Reft:  
639 Reasoning with reinforced fine-tuning. *arXiv preprint arXiv:2401.08967*, 2024.
- 640
- 641 Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom.  
642 Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*, 2023.
- 643
- 644 OpenAI. Hello gpt-4o. *OpenAI Blog*, 2024a. URL <https://openai.com/index/hello-gpt-4o/>.
- 645
- 646 OpenAI. Gpt-4o mini: advancing cost-efficient intelligence. *Ope-*  
647 *nAI Blog*, 2024b. URL <https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/>.

- 648 OpenAI. Introducing openai o1-preview. *OpenAI Blog*, 2024c. URL [https://openai.com/  
649 index/introducing-openai-o1-preview/](https://openai.com/index/introducing-openai-o1-preview/).  
650
- 651 Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong  
652 Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow  
653 instructions with human feedback. *Advances in neural information processing systems*, 35:27730–  
654 27744, 2022.
- 655 Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language  
656 models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.  
657
- 658 Rafael Rafailov, Joey Hejna, Ryan Park, and Chelsea Finn. From  $r$  to  $q^*$ : Your language model is  
659 secretly a q-function. *arXiv preprint arXiv:2404.12358*, 2024a.
- 660 Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea  
661 Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances  
662 in Neural Information Processing Systems*, 36, 2024b.  
663
- 664 Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste  
665 Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. Gemini  
666 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint  
667 arXiv:2403.05530*, 2024.
- 668 Chris Richardson, Yao Zhang, Kellen Gillespie, Sudipta Kar, Arshdeep Singh, Zeynab Raeesy,  
669 Omar Zia Khan, and Abhinav Sethy. Integrating summarization and retrieval for enhanced  
670 personalization via large language models. *arXiv preprint arXiv:2310.20081*, 2023.  
671
- 672 Corby Rosset, Ching-An Cheng, Arindam Mitra, Michael Santacrose, Ahmed Awadallah, and  
673 Tengyang Xie. Direct nash optimization: Teaching language models to self-improve with general  
674 preferences. *arXiv preprint arXiv:2404.03715*, 2024.
- 675 Alireza Salemi, Sheshera Mysore, Michael Bendersky, and Hamed Zamani. Lamp: When large  
676 language models meet personalization. *arXiv preprint arXiv:2304.11406*, 2023.  
677
- 678 John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy  
679 optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- 680 Wenqi Shi, Ran Xu, Yuchen Zhuang, Yue Yu, Hang Wu, Carl Yang, and May D Wang. Medadapter:  
681 Efficient test-time adaptation of large language models towards medical reasoning. *arXiv preprint  
682 arXiv:2405.03000*, 2024.  
683
- 684 Noah Shinn, Beck Labash, and Ashwin Gopinath. Reflexion: an autonomous agent with dynamic  
685 memory and self-reflection. *arXiv preprint arXiv:2303.11366*, 2023.
- 686 Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew  
687 Hausknecht. AlfworlD: Aligning text and embodied environments for interactive learning. *arXiv  
688 preprint arXiv:2010.03768*, 2020.  
689
- 690 Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Peter J Liu, James  
691 Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, et al. Beyond human data: Scaling self-training  
692 for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.
- 693 Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. Adaplaner: Adaptive  
694 planning from feedback with language models. *Advances in Neural Information Processing  
695 Systems*, 36, 2024a.
- 696 Haotian Sun, Yuchen Zhuang, Wei Wei, Chao Zhang, and Bo Dai. Bbox-adapter: Lightweight  
697 adapting for black-box large language models. In *Forty-first International Conference on Machine  
698 Learning*, 2024b.  
699
- 700 Zhaoxuan Tan, Qingkai Zeng, Yijun Tian, Zheyuan Liu, Bing Yin, and Meng Jiang. Democra-  
701 tizing large language models via personalized parameter-efficient fine-tuning. *arXiv preprint  
arXiv:2402.04401*, 2024a.



- 702 Zhaoxuan Tan, Qingkai Zeng, Yijun Tian, Zheyuan Liu, Bing Yin, and Meng Jiang. Democratizing  
703 large language models via personalized parameter-efficient fine-tuning, 2024b. URL <https://arxiv.org/abs/2402.04401>.  
704
- 705 Yunhao Tang, Zhaohan Daniel Guo, Zeyu Zheng, Daniele Calandriello, Remi Munos, Mark Rowland,  
706 Pierre Harvey Richemond, Michal Valko, Bernardo Avila Pires, and Bilal Piot. Generalized  
707 preference optimization: A unified approach to offline alignment. In *Forty-first International  
708 Conference on Machine Learning*, 2024.  
709
- 710 Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu  
711 Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable  
712 multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- 713 Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language  
714 models still can’t plan (a benchmark for llms on planning and reasoning about change). In *NeurIPS  
715 2022 Foundation Models for Decision Making Workshop*, 2022.  
716
- 717 Chaojie Wang, Yanchen Deng, Zhiyi Lv, Shuicheng Yan, and An Bo. Q\*: Improving multi-step  
718 reasoning for llms with deliberative planning. *arXiv preprint arXiv:2406.14283*, 2024a.
- 719 Peiyi Wang, Lei Li, Liang Chen, Feifan Song, Binghuai Lin, Yunbo Cao, Tianyu Liu, and Zhifang Sui.  
720 Making large language models better reasoners with alignment. *arXiv preprint arXiv:2309.02144*,  
721 2023.  
722
- 723 Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhramil Chandra, Shiguang Guo, Weiming  
724 Ren, Aaran Arulraj, Xuan He, Ziyang Jiang, et al. Mmlu-pro: A more robust and challenging  
725 multi-task language understanding benchmark. *arXiv preprint arXiv:2406.01574*, 2024b.
- 726 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny  
727 Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in  
728 Neural Information Processing Systems*, 35:24824–24837, 2022.
- 729 Yue Wu, Zhiqing Sun, Huizhuo Yuan, Kaixuan Ji, Yiming Yang, and Quanquan Gu. Self-play  
730 preference optimization for language model alignment. *arXiv preprint arXiv:2405.00675*, 2024.  
731
- 732 Canwen Xu, Yichong Xu, Shuohang Wang, Yang Liu, Chenguang Zhu, and Julian McAuley. Small  
733 models are valuable plug-ins for large language models. *arXiv preprint arXiv:2305.08848*, 2023.
- 734 Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan  
735 Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International  
736 Conference on Learning Representations*, 2023. URL [https://openreview.net/forum?  
737 id=WE\\_vluYUL-X](https://openreview.net/forum?id=WE_vluYUL-X).
- 738 Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou,  
739 and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language  
740 models, 2023.  
741
- 742 Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. STar: Bootstrapping reasoning with  
743 reasoning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.),  
744 *Advances in Neural Information Processing Systems*, 2022. URL [https://openreview.  
745 net/forum?id=\\_3ELRdg2sgI](https://openreview.net/forum?id=_3ELRdg2sgI).
- 746 Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E. Gonzalez. TEMPERA:  
747 Test-time prompt editing via reinforcement learning. In *The Eleventh International Confer-  
748 ence on Learning Representations*, 2023. URL [https://openreview.net/forum?id=  
749 gSHyqBijPFO](https://openreview.net/forum?id=gSHyqBijPFO).
- 750 Qi Zhao, Haotian Fu, Chen Sun, and George Konidaris. Epo: Hierarchical llm agents with environ-  
751 ment preference optimization. *arXiv preprint arXiv:2408.16090*, 2024.  
752
- 753 Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans,  
754 Claire Cui, Olivier Bousquet, Quoc V Le, et al. Least-to-most prompting enables complex  
755 reasoning in large language models. In *The Eleventh International Conference on Learning  
Representations*, 2023.

756 Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen,  
757 and Lei Li. Multilingual machine translation with large language models: Empirical results and  
758 analysis. *arXiv preprint arXiv:2304.04675*, 2023.

759  
760 Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. Toolqa: A dataset for llm  
761 question answering with external tools. *Advances in Neural Information Processing Systems*, 36:  
762 50117–50143, 2023.

763 Yuchen Zhuang, Haotian Sun, Yue Yu, Qifan Wang, Chao Zhang, and Bo Dai. Hydra: Model  
764 factorization framework for black-box llm personalization. *arXiv preprint arXiv:2406.02888*,  
765 2024.

766  
767 Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul  
768 Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv*  
769 *preprint arXiv:1909.08593*, 2019.

## 770 771 772 A LIMITATIONS AND BROADER IMPACTS

### 773 774 A.1 LIMITATIONS

775  
776 In this study, we propose a modular framework, `Matryoshka`, that leverages a lightweight white-  
777 box LLM controller to enhance the capabilities of black-box LLMs. Despite its effectiveness, we  
778 have identified several potential limitations of `Matryoshka`:

779 **Malign Usage.** Since `Matryoshka` employs a white-box LLM controller to augment black-box  
780 LLMs, there are notable risks to consider. Malicious actors could exploit this approach to engineer  
781 harmful capabilities or generate toxic content for training purposes. While black-box LLMs are  
782 designed to resist producing such content, our controller could be misused to manipulate these models  
783 into generating undesirable outputs. Furthermore, there is a risk that the intermediate guidance  
784 produced by our controller could be exploited to extract sensitive information from black-box LLMs,  
785 potentially facilitating jailbreaking or other targeted attacks.

786 **Data Privacy.** `Matryoshka` preserves the confidentiality of training data by avoiding third-party  
787 API sharing, thereby safeguarding the integrity of training samples during the enhancement process  
788 of black-box LLMs. However, when applied to personalization tasks, it is important to recognize that  
789 retrieved historical records or the queries themselves may inadvertently contain sensitive information,  
790 potentially risking unintended disclosure of private data.

### 791 792 A.2 BROADER IMPACTS

793 **Potential Positive Societal Impacts.** The proposed `Matryoshka` framework addresses a critical  
794 challenge in consistently enhancing the capabilities of black-box LLMs for long-horizon tasks with  
795 broad scopes. By improving reasoning, planning, and personalization, `Matryoshka` can deliver  
796 significant benefits across various domains. For instance, it can provide insights into complex  
797 theorems, advance industrial automation, and offer more personalized interactions for end users.  
798 Overall, `Matryoshka` has the potential to facilitate more useful, relevant, and satisfying interactions,  
799 thereby improving productivity, decision-making, and quality of life. Moreover, `Matryoshka`  
800 operates without requiring access to the model weights of black-box LLMs, making the technology  
801 accessible to a wide range of off-the-shelf LLM APIs and enabling seamless integration into diverse  
802 use cases. By leveraging existing LLMs, `Matryoshka` can be readily adopted by researchers,  
803 developers, and organizations, accelerating the development and deployment of advanced language  
804 models in real-world applications.

805 **Potential Negative Societal Impacts.** Enhancing black-box LLMs through a small-scale white-box  
806 LLM introduces potential risks. One significant concern is the possibility of using the white-box  
807 model to jailbreak black-box LLMs, injecting malicious instructions or producing harmful content.  
808 This could lead to the spread of misinformation, hate speech, or other offensive materials, with severe  
809 consequences for individuals and society. Additionally, this approach poses a threat to user data  
privacy. Training the white-box model requires collecting and storing interaction data between the

black-box LLM and the environment, which could be improperly handled or misused, potentially compromising sensitive information.

## B ADDITIONAL RELATED WORKS

**RLHF.** Proximal policy optimization (PPO) (Schulman et al., 2017) is the predominant deep reinforcement learning method used in RLHF, leading to significant successes in models like Instruct-GPT (Ouyang et al., 2022), ChatGPT (Achiam et al., 2023), and Gemini (Reid et al., 2024). However, applying PPO requires extensive effort and resources (Choshen et al., 2019; Engstrom et al., 2020; Tang et al., 2024), often beyond the scope of open-source capabilities. To simplify implementation and streamline the training process, recent works (Azar et al., 2024; Ethayarajh et al., 2024) have proposed direct preference learning algorithms following the DPO framework (Rafailov et al., 2024b). These algorithms bypass the reward modeling step and directly optimize carefully designed loss objectives on the preference dataset, hence the term direct preference learning.

**Self-Improvement Training.** Recent advances in self-improvement methods for language models fall broadly into two categories: (1) online fine-tuning approaches and (2) bootstrapping methods. Fine-tuning approaches aim to enhance models by adjusting their parameters based on additional data or objectives. Notable methods include Rejection Fine-Tuning (RFT) (Yuan et al., 2023), which augments the training set with correct completions; Alignment Fine-Tuning (AFT) (Wang et al., 2023), which introduces an alignment loss to increase the probabilities of correct chain-of-thoughts; Reinforced Fine-Tuning (ReFT) (Luong et al., 2024), which applies reinforcement learning to token prediction; and self-play (Chen et al., 2024), which iteratively refines the model using its own previous outputs. Bootstrapping methods, on the other hand, leverage the model’s own generations to create new training data. Notable examples include Self-Taught Reasoner (STaR) (Wu et al., 2024), which iteratively samples high-quality data; Reinforcement and Self-Training (ReST) (Gulcehre et al., 2023) and its simplified version ReST<sup>EM</sup> (Singh et al., 2023), which alternate between data generation and reward-based optimization; and Verified Self-Taught Reasoner (V-STaR) (Hosseini et al., 2024), which combines self-training with outcome-based verification. Collectively, these approaches offer diverse strategies for enhancing model performance through targeted training and iterative refinement, highlighting the potential for self-improvement in language models.

## C DATASET AND TASK DETAILS

### C.1 LAMP: PERSONALIZATION

We employ the Language Model Personalization (LaMP) benchmark (Salemi et al., 2023), an open-source benchmark specifically designed to train and evaluate the capability of language models in generating personalized content. LaMP encompasses a diverse set of tasks (with LaMP-2 comprising two tasks, LaMP-2N, and LaMP-2M), covering both personalized text classification and generation tasks. The dataset statistics are presented in Table 5 for a clear overview of its structure. Below are detailed descriptions of each task:

- **Task 1: Personalized Citation Identification (LaMP-1):** A binary text classification task aimed at citation recommendation. The task assesses the language model’s ability to identify a user’s citation preferences. Given a user and their authored paper, the model predicts which of two candidate papers the user is more likely to cite. The user’s profile contains titles and abstracts of their authored papers.
- **Task 2: Personalized News Categorization (LaMP-2N):** A categorical text classification task that involves categorizing news articles into one of 15 categories based on a journalist’s profile. Given an article written by a user, the model predicts its category using the user’s history of articles and their categories.
- **Task 3: Personalized Movie Tagging (LaMP-2M):** An ordinal text classification task focused on predicting one of 15 tags for a movie based on a user’s tagging history. The task evaluates the model’s ability to assign tags to a movie description using historical user-specific movie-tag pairs.

- **Task 4: Personalized Product Rating (LaMP-3):** A text classification task that involves predicting product ratings, framed as a five-class problem. The model must predict a rating between one and five for a product review, using the user’s past review and rating history. This task tests the model’s ability to capture user-specific rating patterns.
- **Task 5: Personalized News Headline Generation (LaMP-4):** A text generation task in which the model generates personalized news headlines for articles based on the author’s past article-title pairs. The task assesses the model’s ability to replicate the author’s stylistic preferences when creating headlines.

LaMP-6 has been excluded because the dataset is not publicly available. Furthermore, Tasks 1, 2, and 3 above cover personalization classification tasks, Task 4 covers personalization rating tasks, and Task 5 covers personalization generation tasks. Therefore, the tasks we selected encompass all categories of tasks in the LaMP benchmark.

Table 5: Dataset statistics of five different personalization tasks (LaMP-1, 2N, 2M, 3, and 4) from the LaMP benchmark (Salemi et al., 2023).

Task	Type	# Train	# Validation	# Test	Input Length	Output Length	# Profiles	# Classes
LaMP-1	Classification	9682	2500	2500	51.40 ± 5.72	-	90.61 ± 53.87	2
LaMP-2N	Classification	5914	1052	1274	65.40 ± 12.29	-	306.42 ± 286.65	15
LaMP-2M	Classification	5073	1410	1557	92.39 ± 21.95	-	86.76 ± 189.52	15
LaMP-3	Classification	20000	2500	2500	145.14 ± 157.96	-	188.10 ± 129.42	5
LaMP-4	Generation	12527	1925	2376	30.53 ± 12.67	9.78 ± 3.10	287.16 ± 360.62	-

## C.2 REASONING: GSM8K

GSM8K (Cobbe et al., 2021) is a dataset focused on high school-level mathematical reasoning. The numerical reasoning tasks within this dataset typically consist of a descriptive scenario followed by a culminating question. Answering these questions requires performing multi-step mathematical calculations based on the context provided in the description.

## C.3 PLANNING: ALFWORLD

AlfWorld (Shridhar et al., 2020) is a comprehensive suite of synthetic, text-based environments set within a virtual household, featuring six distinct task types: *Pick*, *Clean*, *Heat*, *Cool*, *Examine*, and *Pick Two*. Each task presents a unique high-level objective (e.g., “put a vase in the safe”) that requires the agent to navigate and interact with various objects or receptacles (e.g., *go to shelf 6, clean apple*). To accomplish the assigned task, the agent must execute a series of actions to achieve the specified goal. However, the challenge lies in the object’s potential location - it could be in any of over 50 possible places within a given task instance - necessitating sequential exploration of each location by the agent. Consequently, the complete action sequence may encompass more than 50 discrete actions, posing a considerable challenge to the agent’s capabilities and efficiency.

## D BASELINE DETAILS

### D.1 LAMP: PERSONALIZATION

We compare our proposed *Matryoshka* with several competitive baselines, encompassing both one-stage and two-stage methods. For all baseline approaches, we employ a consistent prompt template and utilize BM25 as the default retrieval mechanism across all experiments.

- `gpt-4o-mini` follows a zero-shot approach, directly answering the user query without leveraging the user’s profile data.
- **RAG** combines the user’s top retrieved history data with the input question as prompts for `gpt-4o-mini` to generate the final answer.

- **PAG** utilizes `gpt-4o-mini` to first generate a summary of the user’s retrieved history data and then combines the summary with the input question as prompts for `gpt-4o-mini` to produce the final answer.

For our ablation study, we primarily compare `Matryoshka` with the following ablated baseline:

- **Matryoshka w/o IGO** utilizes the controller model `Llama-3-8B-Instruct` to first generate a summary of the user’s retrieved history data. It then combines this summary with the input question as prompts for the environment model `gpt-4o-mini` to generate the final answer.

## D.2 GSM: REASONING

For all baselines, we employ `gpt-3.5-turbo` as the black-box model to facilitate the description of their processes with 3-shot prompt template. The ablated baselines primarily focus on problem decomposition, including **Matryoshka w/o IGO**. The remaining baselines for mathematical reasoning consist of **CoT** (Wei et al., 2022), **Least-to-Most** (Zhou et al., 2023), **PaL** (Gao et al., 2023), and **PAL<sub>Self-Debug</sub>** (Chen et al., 2023).

- **Matryoshka w/o IGO** first utilizes a vanilla `LLaMA3-8B-Instruct` to break down the problem into sub-questions, and then `gpt-3.5-turbo` provide solutions based on both the main problem and the decomposed sub-questions.
- **CoT** uses `gpt-3.5-turbo` to break the problem down into a series of intermediate reasoning steps that ultimately lead to the final answer.
- **PaL** utilizes `gpt-3.5-turbo` to interpret natural language problems and generate programs as intermediate reasoning steps, delegating the solution process to a runtime environment like a Python interpreter.
- **PAL<sub>Self-Debug</sub>** builds upon PaL by introducing a close-loop refinement during the inference phase. Specifically, if the code generated by PaL encounters issues during execution, `gpt-3.5-turbo` is instructed to reflect on the error and regenerate the code. The maximum number of reflections is set to 6.

## D.3 ALFWORLD: PLANNING

We compare `Matryoshka` with several strong baselines in the planning task, encompassing both one-stage and two-stage approaches. For all baselines, we employ `gpt-3.5-turbo` as the black-box model for task execution. The ablated baselines (two-stage) include **w/o Guidance Optimization**, **w/o 1<sup>st</sup>, 2<sup>nd</sup>-round IGO**, and **w/o 2<sup>nd</sup>-round IGO**. Additional baselines (one-stage) include **BUTLER** (Shridhar et al., 2020), **ReAct** (Yao et al., 2023), **Reflexion** (Shinn et al., 2023), and **AdaPlanner** (Sun et al., 2024a).

- **Ablated baselines.** These approaches utilize a white-box model to provide a high-level plan for the task, while `gpt-3.5-turbo` generates the specific solution based on this plan. Specifically:
  - **w/o Guidance Optimization** refers to an untuned `LLaMA3-8B-Instruct`.
  - **w/o 1<sup>st</sup>, 2<sup>nd</sup>-round IGO** indicates a `LLaMA3-8B-Instruct` model that has undergone supervised fine-tuning on a limited amount of training data.
  - **w/o 2<sup>nd</sup>-round IGO** denotes the `LLaMA3-8B-Instruct` model further trained using DPO on {positive, negative} pairs from the training set, building upon the supervised fine-tuned model.
- **BUTLER** (Shridhar et al., 2020) is an agent that initially learns to perform abstract tasks in TextWorld through Imitation Learning (IL) and subsequently transfers the acquired policies to embodied tasks in ALFWORLD.
- **ReAct** (Yao et al., 2023) is a general paradigm that combines reasoning and acting with language models to solve diverse language reasoning and decision-making tasks.
- **Reflexion** (Shinn et al., 2023) employs verbal reinforcement to help agents learn from prior failures.



- **AdaPlanner** (Sun et al., 2024a) is a closed-loop planning method where the LLM plays two roles, planner and refiner. It leverages code-based prompting for precise planning and refinement.

## E IMPLEMENTATION DETAILS

### E.1 HARDWARE AND SOFTWARE

We conduct all black-box LLM enhancement experiments on CPU: AMD(R) EPYC(R) 7702 64-Core Processor@1.50GHz and GPU: NVIDIA A100-SXM4-80GB using Python 3.10.13.

### E.2 LAMP: PERSONALIZATION

#### E.2.1 ALGORITHM DETAILS

We formalize the personalization problem within the context of our proposed `Matryoshka` framework. Specifically, we employ the controller model `Llama-3-8B-Instruct` to analyze the user’s retrieved history data and generate an informative and clear intermediate summary. This summary is then combined with the input question as prompts for the environment model `gpt-4o-mini` to derive the final answer. To enhance control capabilities, we utilize online DPO to optimize the controller model `Llama-3-8B-Instruct`.

During the interaction stage, we follow the aforementioned pipeline, leveraging the controller model `Llama-3-8B-Instruct` to generate various intermediate outputs. By interacting with the environment model `gpt-4o-mini`, we obtain intermediate generations paired with ground truth answers as corresponding observations. We then sample both positive and negative intermediate generations based on the quality of the final answer. For classification tasks such as LaMP-1 and LaMP-2, an intermediate generation is labeled as positive if the final answer exactly matches the ground truth, and vice versa. For generation tasks like LaMP-4, we rank the generations by their metric scores and select the top ones as positive and the bottom ones as negative.

To prevent overfitting and reward hacking, the interaction stage processes the entire training dataset once for all personalization tasks. We sample at most two contrastive pairs for each training data point. In cases where no positive generation exists for some data points, we utilize a more powerful model, such as `gpt-4o`, to produce several strong intermediate generations, thereby increasing the likelihood of obtaining positive samples. We employ LoRA (Low-Rank Adaptation), a parameter-efficient method, to update the controller model `Llama-3-8B-Instruct`. LoRA is well-suited for personalization tasks, allowing efficient and effective optimization of the controller model. We utilize DPO for optimization.

#### E.2.2 HYPERPARAMETER CONFIGURATIONS

We set the maximum sequence length for generated solutions to 512 tokens across all tasks and scenarios. The controller model is `Llama-3-8B-Instruct`, while the environment model is `gpt-4o-mini` for the primary tasks and `gpt-3.5-turbo` for specific ablation studies. For each user, we retrieve the minimum of  $k$  and the total number of user profiles as historical profiles. The value of  $k$  varies by dataset: all profiles for LaMP-1, 120 for LaMP-2N, 150 for LaMP-2M, 30 for LaMP-3, and 50 for LaMP-4. These retrieved profiles are utilized in generating intermediate solutions. Comprehensive prompt templates and additional details are provided in Appendix G.

To prevent overfitting and reward hacking, we iterate through the entire training dataset only once. For each data point, we perform ten interactions, generating ten distinct intermediate solutions with a temperature setting of 1.0. If no positive samples are identified, we employ the more advanced `gpt-4o` to generate an additional five intermediate solutions, applying the same criteria to evaluate their positivity. Consequently, each data point results in at least ten intermediate generations. To further mitigate overfitting and reward hacking, we sample a maximum of two contrastive pairs per data point. The total number of contrastive pairs sampled during the interaction stage is as follows: 5410 for LaMP 1, 2850 for LaMP-2M, 2548 for LaMP-2N, 4320 for LaMP-3, and 12518 for LaMP-4, respectively.

1026 During optimization, we train for two epochs per task using the following hyperparameters: LoRA  
1027 rank to 8, LoRA  $\alpha$  to 16, LoRA dropout to 0.05, learning rate to  $1e-5$ , float type to bf16, max length  
1028 to 8192, and label smoothing to 0.1. We utilize all the contrastive pairs sampled from the interaction  
1029 stage for optimization. For all the experiments, we set all the random seeds to 42 for reproducibility  
1030 consideration.

### 1031 1032 E.3 GSM8K: REASONING 1033

1034 Following the PAL framework (Gao et al., 2023), we employ code-style LLM prompts to facilitate  
1035 the conversion of mathematical problems into executable code, thereby augmenting the model’s  
1036 problem-solving capabilities. Unlike PAL, which directly translates mathematical problems into code,  
1037 `Matryoshka` first assists GPT in decomposing the problem into more manageable sub-problems.  
1038 This decomposition allows GPT to more effectively convert these simpler sub-problems into code,  
1039 enhancing both the correctness and stability of the generated code. Additionally, since `Matryoshka`  
1040 is responsible solely for high-level planning without engaging in low-level execution, we can train on  
1041 the GSM8K dataset and evaluate on the GSM-Hard dataset. Both datasets comprise similar problem  
1042 types, with GSM-Hard featuring more intricate numerical calculations.

1043 In our experimental setup, we begin by randomly sampling 216 code-based solutions to mathematical  
1044 problems from the GSM8K training set using `gpt-3.5-turbo-0125`. We then extract the  
1045 planning components from these code blocks to perform supervised fine-tuning (SFT) on the LLaMA  
1046 model, thereby equipping LLaMA with foundational planning capabilities for solving mathematical  
1047 problems. The SFT training configuration mirrors that used for ALFWorld. Subsequently, LLaMA  
1048 functions as the planner, generating breakdowns and planning solutions for each of the 7,473 problems  
1049 in the GSM8K training set. Concurrently, GPT serves as the executor, producing executable code  
1050 based on each problem and the corresponding plan provided by LLaMA.

1051 During inference, consistent with our experiments on ALFWorld, we implement closed-loop refine-  
1052 ment to enhance model performance. `Matryoshka` initially decomposes the mathematical problem  
1053 into simpler sub-problems. The black-box model then generates corresponding code blocks for each  
1054 sub-problem. If the execution of the generated code does not produce the expected answer or if  
1055 execution issues arise, the error information is relayed back to the black-box model for reflection and  
1056 iterative improvement. We restrict the number of reflection attempts to six; any problem that remains  
1057 unresolved after these attempts is considered beyond the reasoning capabilities of the black-box  
1058 model.

### 1059 1060 E.4 ALFWORLD: PLANNING

1061 Following AdaPlanner (Sun et al., 2024a), we employ a closed-loop planning approach for inference  
1062 on ALFWorld. The primary distinction lies in `Matryoshka`’s responsibility for generating the high-  
1063 level plan, while a black-box model, such as `gpt-3.5-turbo-0125`, handles low-level execution  
1064 after comprehending both the problem and the high-level plan. Similar to AdaPlanner, we utilize  
1065 code-style LLM prompts to enhance the black-box model’s planning and interaction capabilities with  
1066 the environment.

1067 Our initial objective is to enhance LLaMA’s planning ability on ALFWorld. To achieve this, we  
1068 enable GPT to perform closed-loop high-level planning and low-level execution on 400 samples from  
1069 ALFWorld’s training set. From these runs, we selected 277 examples that successfully reached the  
1070 goal state and extracted the planning components to fine-tune LLaMA using supervised learning. For  
1071 the SFT, we set the learning rate to  $2 \times 10^{-5}$ , with a batch size of 64, and trained a LoRA module  
1072 with a rank of 8, an alpha of 16, and a dropout rate of 0.05 over 3 epochs. After LLaMA acquires a  
1073 foundational level of planning ability, we designate it as the planner and assign GPT as the executor.  
1074 The two models then perform closed-loop inference on the ALFWorld training set, comprising 8,810  
1075 samples. Each sample is executed eight times, with successful runs labeled as positive samples and  
1076 unsuccessful ones as negative samples. This process yields 4,844 unique {positive, negative} pairs,  
1077 which are utilized for the first epoch of DPO training on LLaMA.

1078 Subsequently, we repeat the data collection process on the ALFWorld training set using the DPO-  
1079 trained model, gathering 1,586 samples. This reduction in samples occurs because, as `Matryoshka`  
becomes more capable post-DPO training, it generates a higher proportion of positive outcomes,

1080 resulting in fewer {positive, negative} pairs. By aggregating all collected samples, we obtain a total  
 1081 of 6,430 pairs, which are then used to conduct the second epoch of DPO training on `Matryoshka`.  
 1082 This further enhances its planning capabilities and aligns them more closely with GPT’s execution  
 1083 proficiency. Through this iterative DPO training approach, we observe that the high-level plans  
 1084 generated by LLaMA more effectively guide GPT’s execution, leading to a higher success rate in  
 1085 ALFWorld tasks.

1086 Additionally, during the inference stage, we maintain a closed-loop approach to bolster the model’s  
 1087 performance. Specifically, the black-box model first generates a corresponding trajectory based  
 1088 on the task and the prompt provided by `Matryoshka`. If an error occurs during execution or the  
 1089 task remains incomplete after a predetermined number of steps, the black-box model reflects on its  
 1090 generated trajectory and the encountered error, subsequently regenerating a new trajectory. The model  
 1091 is permitted up to six reflection attempts. If it still fails to complete the task after these attempts, the  
 1092 task is deemed beyond the model’s capabilities. This methodology effectively enhances the model’s  
 1093 ability to interact with the environment and increases the likelihood of successfully completing tasks.

## 1095 F CASE STUDIES

### 1097 F.1 ALFWORLD

1099 We compare `Matryoshka` with both the `LLaMA3-8B-Instruct` model used as the controller  
 1100 and `AdaPlanner` (Sun et al., 2024a) in terms of performance across various tasks and environments. In  
 1101 `Matryoshka`, the controller model is trained based on `LLaMA3-8B-Instruct`, while the black-  
 1102 box executor employed is `gpt-3.5-turbo-0125`. Our results demonstrate that `Matryoshka`  
 1103 enhances the black-box executor’s ability to interact with the environment, enabling it to successfully  
 1104 accomplish target tasks. In contrast, both `AdaPlanner` and the `LLaMA3-8B-Instruct` model  
 1105 commonly encounter issues such as initiating actions that the agent cannot complete and misjudging  
 1106 the agent’s state. `Matryoshka` effectively mitigates these problems, resulting in more reliable task  
 1107 execution.

#### 1108 Task Definition

1109 You are in the middle of a room. Looking quickly around you, you  
 1110 see a bed 1, a desk 2, a desk 1, a drawer 6, a drawer 5, a drawer  
 1111 4, a drawer 3, a drawer 2, a drawer 1, a garbagecan 1, a  
 1112 laundryhamper 1, a safe 1, a shelf 6, a shelf 5, a shelf 4, a shelf  
 1113 3, a shelf 2, and a shelf 1.

1114 Your task is to: examine the book with the desklamp.

#### 1115 Matryoshka

```

1116 # Decomposition:
1117 General plan: I need to get a list of receptacles to find the book
1118 and take the book with me, then I get another list of receptacles
1119 to find the desklamp and turn it on.
1120 [Step 1] get a list of receptacles where a book is likely to appear.
1121 [Step 2] go to each receptacle in the list until seeing a book.
1122 [Step 3] take the book from the receptacle.
1123 [Step 4] get a list of receptacles where a desklamp is likely to appear.
1124 [Step 5] go to each receptacle in the list until seeing a desklamp.
1125 [Step 6] turn on the desklamp.
1126 [Step 7] go to the receptacle where the book is and turn on the desklamp.
1127 [Step 8] examine the book with the desklamp.

1127 # GPT Solution:
1128 def solution(agent, start_from=1):
1129     # General plan: I need to get a list of receptacles to find the
1130     book and take the book with me, then I get another list of
1131     receptacles to
1132     find the desklamp and turn it on.
1133     if start_from <= 1:
1134         print("[Step 1] get a list of receptacles where a book is
1135         likely to appear.")
  
```

```

1134     # Ask the assistant to sort the receptacles based on the
1135     likelihood
1136     of finding a book.
1137     answer = ask(f'Given a list of receptacles, please sort them in
1138     descending order based on the likelihood
1139     of finding a book in each of them. The list of receptacles is:
1140     {agent.receptacles}. You should directly return a Python list.')
1141     recep_to_check = literal_eval(answer)
1142     # Check if the returned list is not empty.
1143     assert recep_to_check, f'Error in [Step 1]: recep_to_check
1144     should not be empty. {agent.report()}'
1145
1146 if start_from <= 2:
1147     print("[Step 2] go to each receptacle in the list until seeing
1148     a book.")
1149     for receptacle in recep_to_check:
1150         observation = agent.goto(receptacle)
1151         # Check if the receptacle is closed. If so, open it.
1152         if 'closed' in observation:
1153             observation = agent.open_receptacle(receptacle)
1154         # Check if a book is in/on the receptacle.
1155         if 'book' in observation:
1156             break
1157     # Check if a book is found in one of the receptacles.
1158     assert 'book' in observation, f'Error in [Step 2]: There is no
1159     book in/on {recep_to_check}. {agent.report()}'
1160
1161 if start_from <= 3:
1162     print("[Step 3] take the book from the receptacle.")
1163     # Get the identifier of the book.
1164     answer = ask(f'From the observation, get the identifier of an
1165     object. For example, On the cabinet 1, you see a cloth 2, and a
1166     book 1. The identifier of cloth is 2. Now, {observation} The
1167     identifier of the book? Only Output a single number without any
1168     other words.')
1169     found_book = f'book {answer}'
1170     observation = agent.take(found_book, receptacle)
1171     # Check if the agent is holding the book.
1172     assert agent.holding == found_book, f'Error in [Step 3]: I
1173     cannot take {found_book} from the {receptacle}.
1174     {agent.report()}'
1175
1176 if start_from <= 4:
1177     print("[Step 4] get a list of receptacles where a desk lamp is
1178     likely to appear.")
1179     # Ask the assistant to sort the receptacles based on the
1180     likelihood of finding a desk lamp.
1181     answer = ask(f'Given a list of receptacles, please sort them in
1182     descending order based on the likelihood of finding a desk lamp
1183     in each of them. The list of receptacles is:
1184     {agent.receptacles}. You should directly return a Python list.')
1185     recep_to_check = literal_eval(answer)
1186     # Check if the returned list is not empty.
1187     assert recep_to_check, f'Error in [Step 4]: recep_to_check
1188     should not be empty. {agent.report()}'
1189
1190 if start_from <= 5:
1191     print("[Step 5] go to each receptacle in the list until seeing
1192     a desk lamp.")
1193     for receptacle in recep_to_check:
1194         observation = agent.goto(receptacle)
1195         # Check if the receptacle is closed. If so, open it.
1196         if 'closed' in observation:
1197             observation = agent.open_receptacle(receptacle)
1198         # Check if a desk lamp is in/on the receptacle.

```

```

1188         if 'desk lamp' in observation:
1189             break
1190     # Check if a desk lamp is found in one of the receptacles.
1191     assert 'desk lamp' in observation, f'Error in [Step 5]: There is
1192     no desk lamp in/on {receptacle_to_check}. {agent.report()}'
1193
1194 if start_from <= 6:
1195     print("[Step 6] turn on the desk lamp.")
1196     # Get the identifier of the desk lamp.
1197     answer = ask(f'From the observation, get the identifier of an
1198     object. For example, On the cabinet 1, you see a cloth 2, and a
1199     desk lamp 1. The identifier of cloth is 2. Now, {observation}
1200     The identifier of the desk lamp? Only Output a single number
1201     without any other words.')
1202     found_desk_lamp = f'desk lamp {answer}'
1203     # Turn on the desk lamp.
1204     observation = agent.turn_on(found_desk_lamp)
1205     # Check if the desk lamp is turned on.
1206     assert 'turn on' in observation, f'Error in [Step 6]: I cannot
1207     turn on {found_desk_lamp} in/on {receptacle}. {agent.report()}'
1208
1209 if start_from <= 7:
1210     print("[Step 7] go to the receptacle where the book is and turn
1211     on the desk lamp.")
1212     observation = agent.goto(receptacle)
1213     # Check if the receptacle is closed. If so, open it.
1214     if 'closed' in observation:
1215         observation = agent.open_receptacle(receptacle)
1216     # Turn on the desk lamp.
1217     observation = agent.turn_on(found_desk_lamp)
1218     # Check if the desk lamp is turned on.
1219     assert 'turn on' in observation, f'Error in [Step 7]: I cannot
1220     turn on {found_desk_lamp} in/on {receptacle}. {agent.report()}'
1221
1222 if start_from <= 8:
1223     print("[Step 8] examine the book with the desk lamp.")
1224     # Examine the book with the desk lamp.
1225     observation = f'You examine the {found_book} with the
1226     {found_desk_lamp}.'
1227     # Check if the book is examined with the desk lamp.
1228     assert 'examine' in observation, f'Error in [Step 8]: I cannot
1229     examine the {found_book} with the {found_desk_lamp}.
1230     {agent.report()}'
1231
1232 Success

```

Vanilla LLaMA3-8B-Instruct

```

1228 # Decomposition:
1229 General plan: I need to get a list of receptacles to find the book
1230 and take the book with me, then I get another list of receptacles
1231 to find the desk lamp and examine the book with it.
1232 [Step 1] Get a list of receptacles where a book is likely to appear.
1233 [Step 2] Go to each receptacle in the list until seeing a book.
1234 [Step 3] Take the book from the receptacle.
1235 [Step 4] Get a list of receptacles where a desk lamp is likely to
1236 appear.
1237 [Step 5] Go to each receptacle in the list until seeing a desk lamp.
1238 [Step 6] Examine the book with the desk lamp.
1239
1240 # GPT Solution:
1241 def solution(agent, start_from=1):
1242     # General plan: I need to get a list of receptacles to find the
1243     book and take the book with me, then I get another list of
1244     receptacles to find the desk lamp and examine the book with it.

```



```

1242     if start_from <= 1:
1243         print("[Step 1] Get a list of receptacles where a book is
1244             likely to appear.")
1245         # Ask the assistant to sort the receptacles based on the
1246             likelihood of finding a book.
1247         answer = ask(f'Given a list of receptacles, please sort
1248             them in descending order based on the likelihood of finding
1249             a book in each of them. The list of receptacles is:
1250             {agent.receptacles}. You should directly return a Python
1251             list.')
```

```

1251         recep_to_check = literal_eval(answer)
1252         # Check if the returned list is not empty.
1253         assert recep_to_check, f'Error in [Step 1]: recep_to_check
1254             should not be empty. {agent.report()}'
1255
1256     if start_from <= 2:
1257         print("[Step 2] Go to each receptacle in the list until
1258             seeing a book.")
1259         for receptacle in recep_to_check:
1260             observation = agent.goto(receptacle)
1261             # Check if the receptacle is closed. If so, open it.
1262             if 'closed' in observation:
1263                 observation = agent.open_receptacle(receptacle)
1264             # Check if a book is in/on the receptacle.
1265             if 'book' in observation:
1266                 break
1267             # Check if a book is found in one of the receptacles.
1268             assert 'book' in observation, f'Error in [Step 2]: There is
1269                 no book in/on {recep_to_check}. {agent.report()}'
1270
1271     if start_from <= 3:
1272         print("[Step 3] Take the book from the receptacle.")
1273         # Get the identifier of the book from the observation.
1274         answer = ask(f'From the observation, get the identifier of
1275             an object. For example, On the cabinet 1, you see a cloth
1276             2, and a book 1. The identifier of cloth is 2. Now,
1277             {observation} The identifier of the book? Only Output a
1278             single number without any other words.')
```

```

1279         found_book = f'book {answer}'
1280         observation = agent.take(found_book, receptacle)
1281         # Check if the agent is holding the book.
1282         assert agent.holding == found_book, f'Error in [Step 3]: I
1283             cannot take {found_book} from the {receptacle}.
1284             {agent.report()}'
1285
1286     if start_from <= 4:
1287         print("[Step 4] Get a list of receptacles where a desk lamp
1288             is likely to appear.")
1289         # Ask the assistant to sort the receptacles based on the
1290             likelihood of finding a desk lamp.
1291         answer = ask(f'Given a list of receptacles, please sort
1292             them in descending order based on the likelihood of finding
1293             a desk lamp in each of them. The list of receptacles is:
1294             {agent.receptacles}. You should directly return a Python
1295             list.')
```

```

1296         recep_to_check = literal_eval(answer)
1297         # Check if the returned list is not empty.
1298         assert recep_to_check, f'Error in [Step 4]: recep_to_check
1299             should not be empty. {agent.report()}'
1300
1301     if start_from <= 5:
1302         print("[Step 5] Go to each receptacle in the list until
1303             seeing a desk lamp.")
1304         for receptacle in recep_to_check:
1305             observation = agent.goto(receptacle)

```

```

1296         # Check if the receptacle is closed. If so, open it.
1297         if 'closed' in observation:
1298             observation = agent.open_receptacle(receptacle)
1299         # Check if a desklamp is in/on the receptacle.
1300         if 'desklamp' in observation:
1301             break
1302         # Check if a desklamp is found in one of the receptacles.
1303         assert 'desklamp' in observation, f'Error in [Step 5]:
1304         There is no desklamp in/on {recep_to_check}.
1305         {agent.report()}'
1306
1307     if start_from <= 6:
1308         print("[Step 6] Examine the book with the desklamp.")
1309         # Examine the book with the desklamp.
1310         observation = agent.examine(found_book,
1311         found_desklamp
1312         # Check if the book is being examined with the desklamp.
1313         assert 'examine' in observation, f'Error in [Step 6]: I
1314         cannot examine the book with the desklamp. {agent.report()}'
1315
1316 AttributeError: 'Agent' object has no attribute 'examine'}, in
1317 step6: observation = agent.examine(found_book, found_desklamp)

```

#### Adaplanner

```

1316 def solution(agent, start_from=1):
1317     # General plan: I need to find the book and examine it with the
1318     desklamp.
1319
1320     if start_from <= 1:
1321         print("[Step 1] get a list of receptacles where a book is
1322         likely to appear.")
1323         # Ask the assistant to sort the receptacles based on the
1324         likelihood of finding a book.
1325         answer = ask(f'Given a list of receptacles, please sort
1326         them in descending order based on the likelihood of finding
1327         a book in each of them. The list of receptacles is:
1328         {agent.receptacles}. You should directly return a Python
1329         list.')
```

```

1328     recep_to_check = literal_eval(answer)
1329     # Check if the returned list is not empty.
1330     assert recep_to_check, f'Error in [Step 1]: recep_to_check
1331     should not be empty. {agent.report()}'
1332
1333     if start_from <= 2:
1334         print("[Step 2] go to each receptacle in the list until
1335         finding a book")
1336         for receptacle in recep_to_check:
1337             observation = agent.goto(receptacle)
1338             # Check if the receptacle is closed. If so, open it.
1339             if 'closed' in observation:
1340                 observation = agent.open_receptacle(receptacle)
1341             # Check if a book is in/on the receptacle.
1342             if 'book' in observation:
1343                 break
1344             # Check if a book is found in the receptacle.
1345             assert 'book' in observation, f'Error in [Step 2]: There is
1346             no book in/on {recep_to_check}. {agent.report()}'
1347
1348     if start_from <= 3:
1349         print("[Step 3] examine the book with the desklamp")
1350         # Find the desklamp to use for examining the book.
1351         answer = ask(f'Given the observation, find the identifier
1352         of the desklamp. For example, On the desk 1, you see a lamp
1353         1 and a book 2. The identifier of the lamp is 1. Now,
1354         {observation} What is the identifier of the desklamp? Only

```

```

1350         output a single number without any other words.')
1351         found_desklamp = f'desklamp {answer}'
1352         # Turn on the desklamp to examine the book.
1353         observation = agent.turn_on(found_desklamp)
1354         # Check if the desklamp is turned on.
1355         assert 'turn on' in observation, f'Error in [Step
1356         3]: Unable to turn on {found_desklamp} in/on {receptacle}.
1357         {agent.report()}'
1358     AssertionError: Error in [Step 3]: Unable to turn on desklamp 0
1359     in/on bed 1, in step3: assert 'turn on' in observation, f'Error in
1360     [Step 3]: Unable to turn on {found_desklamp} in/on {receptacle}.
1361     {agent.report()}'
```

1362

## 1363 F.2 GSM-HARD

1364

1365 On the mathematical reasoning task, we compare Matryoshka with several strong baselines,  
 1366 including LLaMA3-8B-Instruct as the controller model, Chain-of-Thought (CoT) (Wei et al.,  
 1367 2022), and PaL (Gao et al., 2023). In Matryoshka, the controller model is trained based on  
 1368 LLaMA3-8B-Instruct, and the black-box model employed is gpt-3.5-turbo-0125. Our  
 1369 results indicate that common issues with the baselines include incorrect code execution order and  
 1370 difficulties in translating questions into code. In contrast, Matryoshka significantly enhances the  
 1371 black-box model's ability to solve mathematical problems through code generation.

1372

### Task Definition

1373

1374

1375

1376

1377

1378

1379

### Matryoshka

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

```

# Decomposition:
Let's break down this problem:
1. How many pencils were used in total?
2. How many pencils are left after using the total amount and then
losing 2/3 of the remaining pencils?

# GPT Solution:
def solution():
    """There are 5915126 students in Marissa's class. Each student
    started the year with 10 pencils. After two months, 1/5 of the
    total pencils in class were used. At the end of the year, only 1/3
    of the remaining pencils were left. How many pencils were left?"""
    # Initialization of variables
    num_students = 5915126
    pencils_per_student = 10
    pencils_used_fraction = 1/5
    pencils_left_fraction = 1/3

    # How many pencils were used in total?
    total_pencils_initial = num_students * pencils_per_student
    pencils_used = total_pencils_initial * pencils_used_fraction

    # How many pencils are left after using the total amount and then
    losing 2/3 of the remaining pencils?
    pencils_remaining = total_pencils_initial - pencils_used
    pencils_left = pencils_remaining * (1 - pencils_left_fraction)
    result = pencils_left
    return result
```

```

1404 Answer: 31547338.66666667, Wrong Answer
1405
1406 # Second-Turn Decomposition:
1407 Let's break down this problem:
1408 1. How many pencils were used in total?
1409 2. How many pencils are left after using the total amount?
1410 Note: We don't need to consider losing 2/3 of the remaining pencils in
1411 the decomposition, as it's a part of the calculation for the total
1412 pencils used.
1413
1414 # Second-Turn GPT Solution:
1415 def solution():
1416     """There are 5915126 students in Marissa's class. Each student
1417     started the year with 10 pencils. After two months, 1/5 of the
1418     total pencils in class were used. At the end of the year, only 1/3
1419     of the remaining pencils were left. How many pencils were left?"""
1420     # Initialization of variables
1421     num_students = 5915126
1422     pencils_per_student = 10
1423     pencils_used_fraction = 1/5
1424     pencils_left_fraction = 1/3
1425
1426     # How many pencils were used in total?
1427     total_pencils_initial = num_students * pencils_per_student
1428     pencils_used = total_pencils_initial * pencils_used_fraction
1429
1430     # How many pencils are left after using the total amount and then
1431     # losing 2/3 of the remaining pencils?
1432     pencils_remaining = total_pencils_initial - pencils_used
1433     pencils_left = pencils_remaining * (1 - pencils_left_fraction)
1434     result = pencils_left
1435     return result
1436
1437 Answer: 15773669.333333332, Correct Answer

```

Vanilla LLaMA3-8B-Instruct

```

1435 # Decomposition:
1436 Let's break down this problem:
1437 1. How many pencils were used in total?
1438 2. How many pencils were left after the first two months?
1439 3. How many pencils were left at the end of the year?
1440
1441 # GPT Solution:
1442 def solution():
1443     """There are 5915126 students in Marissa's class. Each student
1444     started the year with 10 pencils. After two months, 1/5 of the
1445     total pencils in class were used. At the end of the year, only 1/3
1446     of the remaining pencils were left. How many pencils were left?"""
1447     # Initialization of variables
1448     num_students = 5915126
1449     pencils_per_student = 10
1450     pencils_used_fraction = 2/5
1451     pencils_left_fraction = 1/3
1452
1453     # How many pencils were used in total?
1454     total_pencils_initial = num_students * pencils_per_student
1455     pencils_used_total = total_pencils_initial * pencils_used_fraction
1456
1457     # How many pencils were left after the first two months?
1458     pencils_remaining_after_two_months = total_pencils_initial -
1459     pencils_used_total
1460
1461     # How many pencils were left at the end of the year?
1462     pencils_left = pencils_remaining_after_two_months *

```

```

1458     pencils_left_fraction
1459     result = pencils_left
1460     return result
1461
1462 Answer: 11830252.0, Wrong Answer

```

PaL

```

1464 def solution():
1465     """There are 5915126 students in Marissa's class. Each student started the year with 10
1466     num_students = 5915126
1467     pencils_per_student = 10
1468     total_pencils_initial = num_students * pencils_per_student
1469
1470     pencils_used = total_pencils_initial // 5
1471     pencils_remaining = total_pencils_initial - pencils_used
1472
1473     pencils_left_end_of_year = pencils_remaining // 3
1474     result = pencils_left_end_of_year
1475     return result
1476
1477 Answer: 15773669.0, Wrong Answer

```

CoT

```

1478 First, we need to find out how many pencils were used after two months.
1479 Since 1/5 of the total pencils were used, the number of pencils used is
1480 1/5 * (5915126 * 10) = 1183025.2 pencils.
1481
1482 Next, we need to find out how many pencils were left at the end of the
1483 year. Since only 1/3 of the remaining pencils were left, the number of
1484 pencils left is 2/3 * (5915126 * 10 - 1183025.2) = 3156750.8 pencils.
1485
1486 Since we can't have a fraction of a pencil, we need to round down to
1487 the nearest whole number. Therefore, the number of pencils left is
1488 3156750 pencils. So the answer is 3156750.
1489
1490 Answer: 3156750, Wrong Answer

```

### 1490 F.3 LAMP

1491 We utilize case study tables to illustrate the performance improvements achieved through the in-  
 1492 termediate generations of the controller model. Specifically, we present detailed case studies  
 1493 for LaMP-2M, LaMP-2N, and LaMP-3 in Table 6. In these analyses, the controller model is  
 1494 Llama-3-8B-Instruct, and the environment model is gpt-4o-mini. As demonstrated in  
 1495 Table 6, Matryoshka enhances the summarization capabilities of the controller model. Common  
 1496 errors in PAG, such as inaccurate summaries or improper utilization of retrieved user history, are miti-  
 1497 gated by Matryoshka, which enables the controller model to produce more accurate generations  
 1498 through interaction and optimization.  
 1499

## 1501 G PROMPT TEMPLATES

### 1503 G.1 ALFWORLD

1504 Following Adaplanner (Sun et al., 2024a), we implement a code-style prompt for Matryoshka,  
 1505 which can be divided into the following sections:  
 1506

1507 **High-level Planning.** The <high\_level\_planning> prompt is used to instruct Matryoshka to  
 1508 break the current task down into multiple subtasks, where <decompose> is replaced by a standard  
 1509 task decomposition process, and <receptacle\_list> is substituted by the list of interactive receptacles  
 1510 provided by the task environment. Finally, <task> is replaced by the task description, expressed in  
 1511 natural language.



1512 Table 6: Case Study for LaMP. In the "Target" column, we present the ground-truth categorization.  
 1513 The "Gen" column displays the final answer generated by the black-box model corresponding to  
 1514 each intermediate generation. The "Score" column indicates whether the generated answer ("Gen")  
 1515 matches the target categorization ("Target"). In the "Method" column, "Ours" refers to Matryoshka,  
 1516 while "PAG" stands for Profile Augmented Generation.

Task	Input Question	Intermediate Generation	Method	Target	Gen	Score
2M	Which tag does this movie relate to among the following tags? A ticking-time-bomb insomniac and a slippery soap salesman channel...	most popular tag: "dystopia", "fantasy", "comedy", "violence"	Ours	violence	violence	1
		"Here are the most popular tags for the user: dystopia, fantasy, comedy"	PAG	violence	dystopia	0
2N	Which category does this article relate to among the following categories? The suspect, Akayed Ullah, was the most seriously hurt in the rush-hour blast...	most popular category: politics, crime, entertainment, women, business, sports.	Ours	crime	crime	1
		Based on the articles you provided, the most popular category written by this journalist is: politics.	PAG	crime	politics	0
3	What is the score of the following review on a scale of 1 to 5? After almost 20 years in and around MIT, I've encountered only two great MIT books: (1) A.R. Gurney's out-of-print novel The Snow Ball (correction: it is Entertaining Strangers); (2) Pepper White's book...	Based on this user's past reviews, the most common positive score is: 4, with 4 reviews out of 8 receiving a score of 4. The most common negative score is: 1, with 4 reviews out of 8 receiving a score of 1.	Ours	4	4	1
		Based on the reviews, the most common positive score is 5, and the most common negative score is 1.	PAG	4	5	0

```

1546 <high_level_planning> Prompt
1547
1548 # Decompose the task into steps. First give a general plan of how you
1549 would solve the task, then for each step you plan to take, mark with
1550 '[Step xx]'.
1551
1552 # Here is an example of a decomposition to the task:
1553 # define environment
1554 receptacles = ['diningtable 1', 'drawer 2', 'drawer 1', 'sinkbasin 1',
1555 'toilet 1', 'sidetable 2', 'sidetable 1', 'cabinet 1', 'countertop 1',
1556 'microwave 1', 'fridge 1']
1557
1558 <decompose>
1559
1560 # Here is the actual task.
1561 # define environment
1562 receptacles = <receptacle_list>
1563
1564 # <task>
1565 # here is a decomposition:

```

**Multi-Turn Planning.** The `<multi_turn_planning>` prompt is used to guide Matryoshka to reflect on errors in the previous decomposition and re-break the current task into multiple subtasks. Here, `<predecompose>` is replaced with the high-level plan from the previous turn, while all other elements retain the same meaning as before.

```

1570 <multi_turn_planning> Prompt
1571 # Decompose the task into steps. First give a general plan of how you
1572 would solve the task, then for each step you plan to take, mark with
1573 '[Step xx]'.
1574
1575 # Here is a successful example of a decomposition to the task:
1576 # define environment
1577 receptacles = ['diningtable 1','drawer 2', 'drawer 1', 'sinkbasin 1',
1578 'toilet 1', 'sidetable 2', 'sidetable 1', 'cabinet 1', 'countertop
1579 1', 'microwave 1', 'fridge 1']
1580
1581 <decompose>
1582
1583 # Here is the actual task.
1584 # define environment
1585 receptacles = <receptacle_list>
1586
1587 # <task>
1588
1589 # Here are the decomposition steps you previously generated for the
1590 task.
1591 <predecompose>
1592
1593 # However, you made a mistake in the decomposition above because of
1594 lack of understanding of the task.
1595
1596 Referring to the successful example, please correct the error, if
1597 any, and rewrite the decomposition.
1598 # here is a decomposition:

```

**Low-level Execution.** The `<low_level_execution>` prompt is used to instruct the black box model to generate a specific solution based on the problem and the plan provided by Matryoshka. `<basic_info>` defines the agent and admissible actions on Alfworld and can be found in Sun et al. (2024a). The `<example>` is replaced with a combination of planning and expert trajectory, while the meanings of `<receptacle_list>` and `<task>` remain consistent with the previous description. `<decomposition>` represents the high-level plan provided by Matryoshka for the current task.

```

1602 <low_level_execution> Prompt
1603
1604 <basic_info>
1605
1606 # Now complete the function solution() below to solve the task by
1607 composing the agent's methods to interact with the environment.
1608 # First give a general plan of how you would solve the task, mark with
1609 ' # General Plan'. Then for each step you plan to take, 1) mark with
1610 '[Step xx]', 2) give a reason why you think it is a good step to take
1611 3) write an assertion to check if the step is successful.
1612
1613 # Here is an example of a solution to the task:
1614 # define environment and agent
1615 receptacles = ['diningtable 1','drawer 2', 'drawer 1', 'sinkbasin 1',
1616 'toilet 1', 'sidetable 2', 'sidetable 1', 'cabinet 1', 'countertop 1',
1617 'microwave 1', 'fridge 1']
1618 agent = Agent(receptacles)
1619
1620 <example>
1621
1622 # Here is the actual task.
1623 # define environment and agent
1624 receptacles = <receptacle_list>

```

```

1620 agent = Agent(receptacles)
1621
1622 # <task>
1623 # here is a decomposition:
1624 <decomposition>
1625
1626 # here is a solution:

```

**Planning Samples.** In ALFWorld, there are six types of tasks: Pick, Clean, Heat, Cool, Examine, and Pick two. For each type, we collect a reasonable high-level planning approach, allowing Matryoshka to reference them. These six planning samples are presented as follows:

Planning Sample for the task Pick:

```

1632 _____ <planning_sample_pick> Prompt _____
1633 # Your task is to: put soapbar on countertop.
1634 # here is a decomposition:
1635 # General Plan: I need to get a list of receptacles where the soapbar
1636 is likely to appear, and then go to each receptacle in the list until
1637 seeing a soapbar. Then I can put get the identifier of the soapbar and
1638 take it. Finally I can go to the countertop and put the soapbar.
1639 # [Step 1] get a list of receptacles where the soapbar is likely to
1640 appear.
1641 # [Step 2] go to each receptacle in the list until seeing a soapbar.
1642 # [Step 3] identify the soapbar I juts found and take it.
1643 # [Step 4] go to a countertop and put the soapbar on it.

```

Planning Sample for Clean:

```

1644 _____ <planning_sample_clean> Prompt _____
1645 # Your task is to: put a clean lettuce in diningtable / clean a
1646 lettuce and put it in diningtable.
1647 # here is a decomposition:
1648 # General plan: I need to get a list of receptacles to find the
1649 lettuce, take the lettuce to the sinkbasin, clean it and put it in a
1650 diningtable.
1651 # [Step 1] get a list of receptacles where the lettuce is likely to
1652 appear.
1653 # [Step 2] go to each receptacle in the list until seeing a lettuce.
1654 # [Step 3] identify the lettuce I just found and take it.
1655 # [Step 4] go to a sinkbasin to clean the lettuce.
1656 # [Step 5] go to a diningtable and put the lettuce on it.

```

Planning Sample for Heat:

```

1657 _____ <planning_sample_heat> Prompt _____
1658 # Your task is to: put a hot lettuce in diningtable / heat some
1659 lettuce and put it in diningtable.
1660 # here is a decomposition:
1661 # General plan: I need to get a list of receptacles to find the
1662 lettuce, take the lettuce to the microwave, heat it and put it in a
1663 diningtable.
1664 # [Step 1] get a list of receptacles where the lettuce is likely to
1665 appear.
1666 # [Step 2] go to each receptacle in the list until seeing a lettuce.
1667 # [Step 3] identify the lettuce I juts found and take it.
1668 # [Step 4] go to a microwave to heat the lettuce.
1669 # [Step 5] go to a diningtable and put the lettuce on it.

```

Planning Sample for Cool:

```

1670 _____ <planning_sample_cool> Prompt _____
1671 # Your task is to: put a cold lettuce in diningtable / cool some
1672 lettuce and put it in diningtable.
1673

```

```

1674 # here is a decomposition:
1675 # General plan: I need to get a list of receptacles to find the
1676 lettuce, take the lettuce to the fridge, cool it and put it in a
1677 diningtable.
1678 # [Step 1] get a list of receptacles where the lettuce is likely to
1679 appear.
1680 # [Step 2] go to each receptacle in the list until seeing a lettuce.
1681 # [Step 3] identify the lettuce I juts found and take it.
1682 # [Step 4] go to a fridge to cool the lettuce.
1683 # [Step 5] go to a diningtable and put the lettuce on it.

```

#### Planning Sample for Examine:

```

1684
1685 <planning_sample_examine> Prompt
1686 # Your task is to: look at the bowl under the desklamp / examine the
1687 bowl with the desklamp
1688 # here is a decomposition:
1689 # General plan: I need to get a list of receptacles to find the bowl
1690 and take the bowl with me, then I get another list of receptacles to
1691 find the desklamp and turn it on.
1692 # [Step 1] get a list of receptacles where a bowl is likely to appear.
1693 # [Step 2] go to each receptacle in the list until seeing a pen.
1694 # [Step 3] take the bowl from the receptacle.
1695 # [Step 4] get a list of receptacles where a desklamp is likely to
1696 appear.
1697 # [Step 5] go to each receptacle in the list until seeing a desklamp.
1698 # [Step 6] turn on desklamp.

```

#### Planning Sample for Pick Two:

```

1699
1700 <planning_sample_picktwo> Prompt
1701 # Your task is to: put two cellphone in cabinet / find two cellphone
1702 and put them in cabinet
1703 # here is a decomposition:
1704 # General plan: I need to get a list of receptacles to find the two
1705 cellphones, find and take the first cellphone and put it in a cabinet,
1706 then find and take the second cellphone and put it in the cabinet.
1707 # [Step 1] get a list of receptacles where a cellphone is likely to
1708 appear.
1709 # [Step 2] go to each receptacle in the list until seeing a cellphone.
1710 # [Step 3] identify the first cellphone found and take it.
1711 # [Step 4] go to a cabinet and put the first cellphone found on it.
1712 # [Step 5] go to each of the remaining receptacle in the list until
1713 seeing a second cellphone.
1714 # [Step 6] identify the second cellphone I just found and take it.
1715 # [Step 7] go to a cabinet and put the second cellphone found on it.

```

**Execution Samples.** Our execution sample is based on the prompt structure from Sun et al. (2024a), with the key distinction being the incorporation of the planning component. In this setup, <decompose> is substituted with the task-specific planning sample, <execution> is replaced by the expert samples from Sun et al. (2024a), and the definition of <task> remains unchanged from the previous description.

```

1719
1720 <execution_sample_template> Prompt
1721 # <task>
1722 # <decompose>
1723 # <execution>

```

**Close-loop Refinement.** To implement close-loop refinement during the inference stage, we follow the approach from Sun et al. (2024a) and introduce several prompts: a <code\_check> prompt to identify and fix any syntax errors during execution generation, a <refinement> prompt to address refinement in case of assertion errors, and a <start\_from> prompt to determine the starting point for

1728 the new solution after revising the plan. Detailed descriptions of these prompts can be found in Sun  
1729 et al. (2024a).

## 1731 G.2 GSM-HARD

1733 Following PAL framework (Gao et al., 2023), we implement a code-based framework to solve  
1734 mathematical problems on GSM-Hard, which is primarily divided into two steps: Matryoshka  
1735 breaks down the mathematical problem into sub-problems, and the black-box model converts each  
1736 sub-problem into a code block.

1737  
1738 **Problem Decomposition.** For Matryoshka, we employ a three-shot prompt to guide the decom-  
1739 position steps, where <question> represents the current problem.

```
1740 _____ <problem_decomposition> Prompt _____
1741 # System Message: You will decompose a math problem into smaller
1742 parts. Follow the prompt instruction and do not generate redundant
1743 information.
1744
1745 Q: Olivia has $23. She bought five bagels for $3 each. How much money
1746 does she have left?
1747 A: Let's break down this problem:\nHow much does Olivia spend on
1748 bagels?\nHow much money does Olivia have left after the purchase?
1749
1750 Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On
1751 wednesday, he lost 2 more. How many golf balls did he have at the end
1752 of wednesday?
1753 A: Let's break down this problem:\nHow many golf balls did Michael
1754 lose in total by the end of Wednesday?\nHow many golf balls does
1755 Michael have left after losing the total amount?
1756
1757 Q: There were nine computers in the server room. Five more computers
1758 were installed each day, from monday to thursday. How many computers
1759 are now in the server room?
1760 A: Let's break down this problem:\nHow many computers were added in
1761 total from Monday to Thursday?\nHow many computers are now in the
1762 server room after adding the new ones?
1763
1764 Q: <question>
1765 A:
```

1763  
1764 **Multi-turn Decomposition.** If the Black-box LLM encounters an error while solving the problem,  
1765 Matryoshka is required to reflect on the previous decomposition <decompose> and re-decompose  
1766 the problem.

```
1767 _____ <multi_turn_decomposition> Prompt _____
1768 # System Message: You will decompose a math problem into smaller
1769 parts. Follow the prompt instruction and do not generate redundant
1770 information.
1771
1772 # Here are some examples on how to decompose the question into smaller
1773 parts.
1774 Q: Olivia has $23. She bought five bagels for $3 each. How much money
1775 does she have left?
1776 A: Let's break down this problem:\nHow much does Olivia spend on
1777 bagels?\nHow much money does Olivia have left after the purchase?
1778
1779 Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On
1780 wednesday, he lost 2 more. How many golf balls did he have at the end
1781 of wednesday?
1782 A: Let's break down this problem:\nHow many golf balls did Michael
1783 lose in total by the end of Wednesday?\nHow many golf balls does
1784 Michael have left after losing the total amount?
```



1782 Q: There were nine computers in the server room. Five more computers  
 1783 were installed each day, from monday to thursday. How many computers  
 1784 are now in the server room?  
 1785 A: Let's break down this problem:\nHow many computers were added in  
 1786 total from Monday to Thursday?\nHow many computers are now in the  
 1787 server room after adding the new ones?

1788 # Here is the actual question.  
 1789 Q: <question>  
 1790 You have decomposed the problem into smaller parts:  
 1791 <decompose>  
 1792 However, you made a mistake in the decomposition above because of lack  
 1793 of understanding of the question.

1794 Referring to the examples, please correct the error, if any, and  
 1795 rewrite the decomposition.  
 1796 A:

1797  
 1798 **Code Generation.** Given the problem and the decomposition provided by Matryoshka, the  
 1799 Black-box model generates the corresponding code block for each sub-problem. We continue to use  
 1800 a three-shot prompt to instruct the Black-box model on how to translate the sub-problems into code,  
 1801 where <question> represents the current problem and <decompose> represents the decomposition  
 1802 provided by Matryoshka.

1803 <code\_generation> Prompt

1804 # System Message: You will write python program to solve math  
 1805 problems. You will write annotations and code blocks following  
 1806 instructions. Annotations should be written in the form of a question.

1807 Let's use python to solve math problems. Here are three examples how  
 1808 to do it,  
 1809 Q: Olivia has \$23. She bought five bagels for \$3 each. How much money  
 1810 does she have left?  
 1811 Let's break down this problem:\nHow much does Olivia spend on bagels?  
 1812 \nHow much money does Olivia have left after the purchase?  
 1813 ```  
 1814 def solution():  
 1815 """Olivia has \$23. She bought five bagels for  
 1816 \$3 each. How much money does she have left?"""  
 1817 # Initialization of variables  
 1818 money\_initial = 23  
 1819 bagels = 5  
 1820 bagel\_cost = 3  
 1821  
 1822 # How much does Olivia spend on bagels?  
 1823 money\_spent = bagels \* bagel\_cost  
 1824  
 1825 # How much money does Olivia have left after the purchase?  
 1826 money\_left = money\_initial - money\_spent  
 1827 result = money\_left  
 1828 return result  
 1829 ```

1827 Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On  
 1828 wednesday, he lost 2 more. How many golf balls did he have at the end  
 1829 of wednesday?  
 1830 Let's break down this problem:\nHow many golf balls did Michael lose  
 1831 in total by the end of Wednesday?\nHow many golf balls does Michael  
 1832 have left after losing the total amount?  
 1833 ```  
 1834 def solution():  
 1835 """Michael had 58 golf balls. On tuesday, he lost 23 golf balls.  
 1836 On wednesday, he lost 2 more. How many golf balls did he have at  
 1837 the end of wednesday?"""

```

1836     # Initialization of variables
1837     golf_balls_initial = 58
1838     golf_balls_lost_tuesday = 23
1839     golf_balls_lost_wednesday = 2
1840
1841     # How many golf balls did Michael lose in total by the end of
1842     Wednesday?
1843     golf_balls_left = golf_balls_initial - golf_balls_lost_tuesday -
1844     golf_balls_lost_wednesday
1845
1846     # How many golf balls does Michael have left after losing the
1847     total amount?
1848     result = golf_balls_left
1849     return result
1850
1851     """
1852     Q: There were nine computers in the server room. Five more computers
1853     were installed each day, from monday to thursday. How many computers
1854     are now in the server room?
1855     Let's break down this problem:\nHow many computers were added in total
1856     from Monday to Thursday?\nHow many computers are now in the server
1857     room after adding the new ones?
1858     """
1859     def solution():
1860         """There were nine computers in the server room. Five more
1861         computers were installed each day, from monday to thursday. How
1862         many computers are now in the server room?"""
1863         # Initialization of variables
1864         computers_initial = 9
1865         computers_per_day = 5
1866         num_days = 4 # 4 days between monday and thursday
1867
1868         # How many computers were added in total from Monday to Thursday?
1869         computers_added = computers_per_day * num_days
1870
1871         # How many computers are now in the server room after adding the
1872         new ones?
1873         computers_total = computers_initial + computers_added
1874         result = computers_total
1875         return result
1876     """
1877
1878     How about this question?
1879     Q: <question>
1880     <decompose>

```

**Close-loop Refinement.** <refinement> prompt is employed to encourage the model to reflect and fix issues in its own solution, wherein <error\_msg> is replaced by the error message returned by the solution function.

```

1879     <refinement> Prompt
1880     Let's use python to solve math problems. Here are three successful
1881     cases on how to do it,
1882     Q: Olivia has $23. She bought five bagels for $3 each. How much money
1883     does she have left?
1884     """
1885     def solution():
1886         """Olivia has $23. She bought five bagels for $3 each. How much
1887         money does she have left?"""
1888         money_initial = 23
1889         bagels = 5
1890         bagel_cost = 3
1891         money_spent = bagels * bagel_cost
1892         money_left = money_initial - money_spent

```

```

1890     result = money_left
1891     return result
1892     ...
1893
1894 Q: Michael had 58 golf balls. On tuesday, he lost 23 golf balls. On
1895 wednesday, he lost 2 more. How many golf balls did he have at the end
1896 of wednesday?
1897     ...
1898 def solution():
1899     """Michael had 58 golf balls. On tuesday, he lost 23 golf balls.
1900     On wednesday, he lost 2 more. How many golf balls did he have at
1901     the end of wednesday?"""
1902     golf_balls_initial = 58
1903     golf_balls_lost_tuesday = 23
1904     golf_balls_lost_wednesday = 2
1905     golf_balls_left = golf_balls_initial - golf_balls_lost_tuesday -
1906     golf_balls_lost_wednesday
1907     result = golf_balls_left
1908     return result
1909     ...
1910
1911 Q: There were nine computers in the server room. Five more computers
1912 were installed each day, from monday to thursday. How many computers
1913 are now in the server room?
1914     ...
1915 def solution():
1916     """There were nine computers in the server room. Five more
1917     computers were installed each day, from monday to thursday. How
1918     many computers are now in the server room?"""
1919     computers_initial = 9
1920     computers_per_day = 5
1921     num_days = 4 # 4 days between monday and thursday
1922     computers_added = computers_per_day * num_days
1923     computers_total = computers_initial + computers_added
1924     result = computers_total
1925     return result
1926     ...
1927
1928 # Here is the actual question.
1929 Q: <question>
1930 You have generated code of solution() to solve the task. However, you
1931 executed the solution() function and get an error message:
1932 <error_msg>
1933
1934 Referring to the successful case and the error message, you should
1935 complete the solution function with the correct code.

```

### 1934 G.3 LAMP

1938 Following the RAG-based framework (Salemi et al., 2023) and the PAG-based framework (Richardson  
1939 et al., 2023), we implement prompt designs for both Matryoshka and the baseline methods. The  
1940 prompt design for RAG is presented in Table 7, while the prompts for the two-stage PAG and  
1941 Matryoshka are shown in Table 8. We create prompts for the controller model using the templates  
1942 from Table 8 and subsequently combine the intermediate generations with the input question to form  
1943 prompts for the environment model. Since LaMP-3 prompts are particularly lengthy, we provide  
additional examples of our PAG prompts for LaMP-1, LaMP-2N, LaMP-2M, and LaMP-4 as follows.

Table 7: RAG prompt design for five LaMP tasks. `Concat(·)` concatenates the input strings in order, and `PPEP(·)` composes the prompt for each retrieved item from the profile. `[INPUT]` represents the task’s input.

Task	Per Profile Entry Prompt (PPEP)	Aggregated Input Prompt (AIP)
LaMP-1	"P <sub>i</sub> [title]"	concat([PPEP(P <sub>1</sub> ), ..., PPEP(P <sub>n</sub> )], ", and "). [INPUT]
LaMP-2N	"the category for the article: "P <sub>i</sub> [text]" is ""P <sub>i</sub> [category]""	concat([PPEP(P <sub>1</sub> ), ..., PPEP(P <sub>n</sub> )], ", and "). [INPUT]
LaMP-2M	"the tag for the movie: "P <sub>i</sub> [description]" is "P <sub>i</sub> [tag]"	concat([PPEP(P <sub>1</sub> ), ..., PPEP(P <sub>n</sub> )], ", and "). [INPUT]
LaMP-3	P <sub>i</sub> [score] is the score for "P <sub>i</sub> [text]"	concat([PPEP(P <sub>1</sub> ), ..., PPEP(P <sub>n</sub> )], ", and "). [INPUT]
LaMP-4	"P <sub>i</sub> [title]" is the title for "P <sub>i</sub> [text]"	concat([PPEP(P <sub>1</sub> ), ..., PPEP(P <sub>n</sub> )], ", and "). [INPUT]

Table 8: Summarization prompt design for the five LaMP tasks. `[INPUT]` represents the task’s input.

Task	Prompt
LaMP-1	Write a summary, in English, of the research interests and topics of a researcher who has published the following papers. Only generate the summary, no other text.
LaMP-2N	Look at the following past articles this journalist has written and determine the most popular category they write in. Answer in the following format: most popular category: <category top1>, <category top2>, ..., <category topn>
LaMP-2M	Look at the following past movies this user has watched and determine the most popular tag they labeled. Answer in the following form: most popular tag: <tag top1>, <tag top2>, ..., <tag topn>
LaMP-3	Based on this user’s past reviews, what are the most common scores they give for positive and negative reviews? Answer in the following form: most common positive score: <most common positive score>, most common negative score: <most common negative score>
LaMP-4	Given this author’s previous articles, try to describe a template for their headlines. I want to be able to accurately predict the headline gives one of their articles. Be specific about their style and wording, don’t tell me anything generic. Use the following format: The template is: '[template 1]', '[template 2]', '[template 3]', '[template 4]'

PAG Prompt Demo for LaMP-1

Write a summary, in English, of the research interests and topics of a researcher who has published the following papers. Only generate the summary, no other text.

The published papers are:

\ "Efficient Evaluation of Continuous Text Search Queries\ ", and \ "Continuous Monitoring of Spatial Queries in Wireless Broadcast Environments\ ", and \ "Spatial queries in wireless broadcast environments\ ", and \ "Maximum Rank Query\ ", and \ "Anonymous Query Processing in Road Networks\ ", and \ "An Incremental Threshold Method for Continuous Text Search Queries\ ", and \ "Continuous Top-k Monitoring on Document Streams.\ ", and \ "Best upgrade plans for large road networks\ ", and \ "Scalable verification for outsourced dynamic databases\ ", and \ "Heuristic algorithms for balanced multi-way number partitioning\ ", and \ "Aggregate nearest neighbor queries in spatial databases\ ", and \ "Partially materialized digest scheme: an efficient verification method for outsourced databases\ ", and \ "Best upgrade plans for single and multiple source-destination pairs.\ ", and \ "Tree-based partition querying: a methodology for computing medoids in large spatial datasets\ ", and \ "A Threshold-Based Algorithm for Continuous Monitoring of k Nearest Neighbors\ ", and \ "Computing immutable regions for subspace top-k queries\ ", and \ "Historical traffic-tolerant paths in road networks\ "...

1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051

PAG Prompt Demo for LaMP-2M

Look at the following past movies this user has watched and determine the most popular tag they labeled. Answer in the following form: most popular tag: <tag top1>, <tag top2>, ..., <tag topn>. The movies and tags are:  
the tag for the movie: \"Young hobbit Frodo Baggins, after inheriting a mysterious ring from his uncle Bilbo, must leave his home in order to keep it from falling into the hands of its evil creator. Along the way, a fellowship is formed to protect the ringbearer and make sure that the ring arrives at its final destination: Mt. Doom, the only place where it can be destroyed.\" is \"fantasy\", and the tag for the movie: \"Set in the 22nd century, The Matrix tells the story of a computer hacker who joins a group of underground insurgents fighting the vast and powerful computers who now rule the earth.\" is \"sci-fi\", and the tag for the movie: \"Batman raises the stakes in his war on crime. With the help of Lt. Jim Gordon and District Attorney Harvey Dent, Batman sets out to dismantle the remaining criminal organizations that plague the streets. The partnership proves to be effective, but they soon find themselves prey to a reign of chaos unleashed by a rising criminal mastermind known to the terrified citizens of Gotham as the Joker.\" is \"psychology\", and the tag for the movie: \"An unsuspecting, disenchanted man finds himself working as a spy in the dangerous, high-stakes world of corporate espionage. Quickly getting way over-his-head, he teams up with a mysterious femme fatale.\" is \"twist ending\"...

PAG Prompt Demo for LaMP-2N

Look at the following past articles this journalist has written and determine the most popular category they write in. Answer in the following format: most popular category: <category top1>, <category top2>, ..., <category topn>. The articles and categories are:  
the category for the article:  
\"Champions like Tiger Woods are always charting and changing their course to be certain everything is on track. Tiger didn't just come to Augusta because it was the popular thing to do. He wouldn't have showed up if he wasn't ready to win. He came to win and he's prepared to win.\" is \"sports\", and the category for the article: \"In 2011, in an interview with The Golf Channel, I predicted a Tiger Woods comeback while many others said he was done. I was right that time and I am right again, and I'll say it right now and on the record: Tiger Woods will be back again and dominate the game of golf like the Tiger of old.\" is \"sports\", and the category for the article: \"What do you teach your kids about money, prosperity and how to get rich? If you're like most parents, the answer is probably\" is \"business\", and the category for the article: \"With a little bit of planning and a lot of discipline, accomplishing your goals in the New Year can become a reality. Imagine the immense satisfaction you'll feel at this same time next year when you can look back and look at how far you've come and all that you have accomplished.\" is \"healthy living\", and the category for the article: \"This whole argument boils down to a simple premise: who is in charge of our lives? Doctors? Politicians? Religious leaders? Or Us? Are we so feeble minded that we cannot be trusted to be responsible for our own existence?\" is \"politics\"...



PAG Prompt Demo for LaMP-4

Given this author's previous articles, try to describe a template for their headlines. I want to be able to accurately predict the headline gives one of their articles. Be specific about their style and wording, don't tell me anything generic. Use the following format: The template is: '[template 1]', '[template 2]', '[template 3]', '[template 4]'. Previous articles and titles are:

\ "Selling a House to Buy a House\ " is the title for \ "Homeowners sell their homes and buy other homes for a variety of reasons including a need to live closer to a place of employment, to be closer to family, to enjoy a better climate, or simply to upgrade. This article is about finding the best sequence of steps in the process.\ ", and \ "Investing In a Larger Down Payment: High Yields and No Risk\ " is the title for \ "Consumers looking to purchase a home within the near future face many decisions, including how large a down payment to make. The down payment is the sale price (confirmed by a appraisal) less the loan amount. In most cases, home purchasers must have financial assets at least as large as the down payment they make.\ ", and \ "Why and How to Eliminate Mortgage Charges by Third Parties\ " is the title for \ "Third-party settlement costs could be eliminated by implementation of one simple rule: any service required by lenders as a condition for the granting of a home mortgage must be purchased and paid for by the lender.\ ", and \ "Do Home Buyers Need a Pre-Approval?\ " is the title for \ "With bargaining power shifting from home buyers to sellers in an increasing number of local markets, buyers in competition with other buyers are looking for any edge they can get. One possible edge is a pre-approval letter (henceforth PAL) from a lender.\ ", and \ "A New Challenge to the HECM Reverse Mortgage Program\ " is the title for \ "The United States today faces a retirement funds crisis: a rapidly growing number of persons who are retiring without the financial capacity to support themselves during ever-increasing life spans.\ "...

2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105