LET LARGE LANGUAGE MODELS FIND THE DATA TO TRAIN THEMSELVES

Anonymous authors
Paper under double-blind review

ABSTRACT

The current iterative development process for large language models (LLMs) is heavily data-centric, relying on human researchers and engineers to manually analyze model performance and determine what data to acquire for further training. However, this human-supervised approach is costly and may fail to identify optimal training signals. Its scalability is further limited as models become increasingly capable and may eventually exceed human intelligence. To address these issues, we propose an automated framework that enables models to autonomously discover and strategically acquire the most valuable training data to enhance their performance. It establishes a self-improving framework where models can invoke APIs to crawl and/or generate tailored datasets from various resources and environments, and retrain themselves. The data selection decisions are shaped by reinforcement feedback signals that reward performance gains while penalizing computational overhead. This formulation incentivizes models to develop self-knowledge about their strengths and areas for improvement in order to efficiently select training data. Empirical results demonstrate that LLMs operating within our framework are able to autonomously and strategically acquire valuable training data to enhance their performance across a variety of skills in 1,000 diverse in-house test tasks and three public benchmarks.

1 INTRODUCTION

Large language models (LLMs) (Zhao et al., 2023) have seen remarkable progress in various domains and tasks with extensive training on massive data. Currently, one of the most important aspects of developing LLMs is *data engineering*, which typically involves teams of researchers and engineers meticulously examining model outputs, identifying shortcomings, and then collecting and curating next-iteration training data from various sources to refine model performance (Achiam et al., 2023; Anil et al., 2023; Dubey et al., 2024). While this method has yielded impressive results, it is not without inherent limitations. The human-driven nature of this process introduces potential biases and inefficiencies, as it relies heavily on the intuition and expertise of the development team (Wang et al., 2023; Sun et al., 2024). Furthermore, as these AI models continue to evolve in complexity and capability, there is a growing concern that they may soon exceed human intelligence in certain domains, which raises questions about the long-term viability and scalability of current development practices (Burns et al., 2024).

In this work, we conceptualize and prototype an ACTIVE DATA SEARCH (ADS) framework that facilitates LLMs to automatically acquire training data from external environments for training themselves, without human supervision. In ADS, existing data collection and curation methods are encapsulated into APIs and the LLMs themselves determine when and how they use these APIs for data acquisition. The APIs may involve a wide range of data crawling, filtering, cleaning, refinement, synthesis, and manual annotation techniques to obtain tailored training data. The data sources are also varied, from raw texts extracted via web searches to documents sourced across various platforms, as well as supervised demonstrations generated by AI assistants.

We achieve the above automatic process through development of an optimizer that generates these APIs sequentially based on the target task. Specifically, the optimizer first analyzes the required capabilities for successful task completion, then strategically invokes appropriate API calls to collect training data for performance improvement. To facilitate the optimizer's decision-making process in obtaining optimal training data, we propose a reinforcement learning strategy for optimizer training, leveraging feedback reward signals from the policy to maximize task performance while minimizing computational costs by iterative rejection sampling (RS) (Bai et al., 2022a) and direct preference optimization (DPO) (Rafailov et al., 2024). This refinement process fosters the development of self-knowledge regarding the model's strengths and weaknesses, enabling more efficient training data discovery and utilization. Compared to recent studies (Lozhkov et al., 2024; Wettig et al., 2024; Zhou et al., 2024a) that focus on improving the quality of training data agnostically, a unique characteristic of our framework is its model- and task-specific nature for data acquisition, enabling more tailored and targeted performance enhancement.

In our experiments, we build on top of two popular open-source LLMs: Qwen-2-7B-Instruct (Yang et al., 2024a) and Gemma-2-9B-Instruct (Riviere et al., 2024), and establish three distinct types of APIs for knowledge *acquisitions*, *utilization*, and *enhancement*. Experimental results across 1,000 in-house test tasks with both reward model (RM) and GPT-4 evaluation demonstrate that the LLM equipped with ADS presents consistent performance improvements, achieving a win rate of more than 80% in RM judgment compared to the initial baseline. Moreover, in three public benchmarks: AlpacaEval 2.0 (Dubois et al., 2024), Arena-Hard (Li et al., 2024), and MT-Bench (Zheng et al., 2024), ADS exhibits generalized performance enhancement, even enabling the relative smaller 7B/9B LLMs to compete with the larger 72B/27B counterparts. It is worth noting that across tasks of varying categories and complexities, ADS typically improves performance in knowledge, reasoning, and challenging tasks. To sum up, we highlight our contributions as follows:

- We pioneer the idea of automating the data search process for training LLMs, a task currently handled by expert human efforts, making a further step towards fully automated self-improving AI systems capable of continuous learning and adaptation.
- We propose the ACTIVE DATA SEARCH (ADS) framework, which encapsulates existing data collection and synthesis methods into APIs, enabling LLMs to generate optimal API calls for efficient data discovery through interactions with diverse environments.
- We showcase the effectiveness of ADS by conducting comprehensive experiments on 1,000 diverse in-house test tasks and three public benchmarks. Our results demonstrate that LLMs equipped with ADS exhibit substantial performance improvements. Furthermore, we provide an in-depth analysis of the underlying factors contributing to the efficacy of ADS.

2 RELATED WORK

LLM as an Optimizer With the progressive advancement of LLMs, LLM-as-an-optimizer (i.e., maximizing a downstream metric of an AI system using LLMs without human intervention) has emerged as a new paradigm, where the input optimization task and the output solution are described in natural language and processed by an LLM. This approach has been pioneered in automatic prompt engineering (Pryzant et al., 2023; Zhou et al., 2023; Wang et al., 2024; Guo et al., 2024; Yang et al., 2024b; Xiao et al., 2024), pipeline optimization of LLM-based agents (Zhang et al., 2024; Khattab et al., 2024; Yuksekgonul et al., 2024; Zhuge et al., 2024; Zhou et al., 2024b), and new algorithm discovery (Liu et al., 2024a; Lu et al., 2024). Our research distinguishes itself from these previous studies as our optimization goal is to enhance the fundamental capabilities of LLMs through automated data collection and curation, which is very important given that data is the (perhaps most) major power engine for today's LLMs. Moreover, existing work simply prompts off-the-shelf LLMs for optimization, which can lead to sub-optimal performance as these LLMs have not been explicitly trained to do so. Conversely, we develop methods to fine-tune the optimizer models.

Synthetic Data Generation Synthetic data, which leverages the generation capability of LLMs for data construction, has become crucial components across various stages in LLMs development (Long et al., 2024), including pre-training (Gunasekar et al., 2023; Li et al., 2023; Ben Allal et al., 2024), supervised fine-tuning (Taori et al., 2023; Mukherjee et al., 2023; Xu et al., 2023), and preference learning (Rafailov et al., 2024; Dubey et al., 2024; Adler et al., 2024). Instead of contributing to the booming spectrum by proposing yet another data synthesis method, we introduce a meta-optimization framework, which aims to integrate different data synthesis techniques and develop the best strategy (our optimizer) to leverage them. More recently, there is also a growing interest in targeted data synthesis for enhancing the model's particular weaknesses (Lee et al., 2024; BAAI, 2024; Cheng et al., 2024). Nevertheless, these approaches require extensive evaluation to spot the weaknesses, which sometimes necessitates expert-level knowledge for test set design and

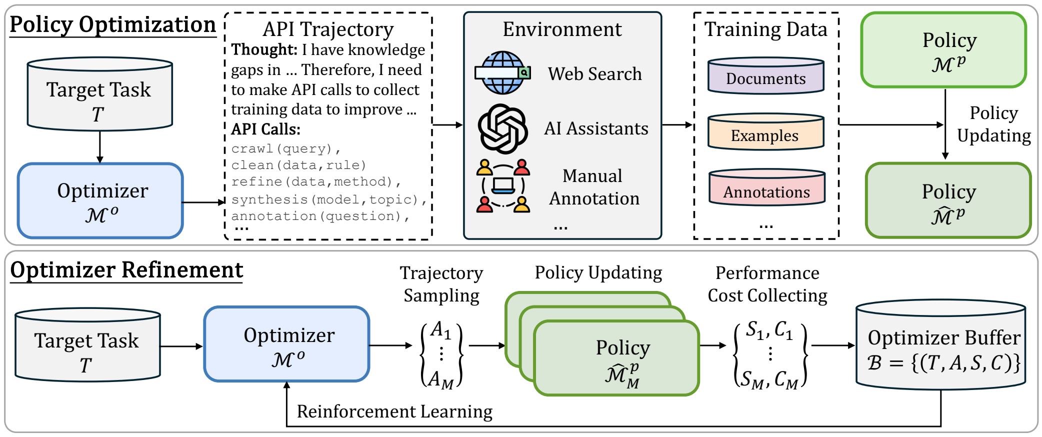

Figure 1: Overview of ACTIVE DATA SEARCH (ADS) framework. Upper: The optimizer is developed to generate optimal API calls and execute them in the environment for training data acquisition. The policy is then updated using the collected training data. Bottom: To make effective and efficient data-collecting decisions, the optimizer is iteratively refined through reinforcement learning.

result analysis. In contrast, our work strives to develop self-knowledge capabilities; the AI system knows its strengths and areas for improvement without any external feedback or guidance. Specifically, given only a small set of representative task queries, the optimizer introspects and identifies potential capability gaps that prevent the policy from completing the target task, and strategically generates optimal API calls to collect training data for addressing the identified gaps.

Self-Improving of LLMs Developing an autonomous self-improving system has been a major focus of current LLM research. Conventional reinforcement learning from human feedback (RLHF) methods (Ziegler et al., 2019; Stiennon et al., 2020; Ouyang et al., 2022; Dong et al., 2024; Liu et al., 2024b; Rosset et al., 2024) train a reward model from human preference data and subsequently use this model to fine-tune LLMs via reinforcement learning. The reliance on human preference data of course can be removed for scenarios where the correctness of output can be validated automatically and unambiguously (Zelikman et al., 2022; Gulcehre et al., 2023; Singh et al., 2023). For general instruction following, recent studies have built upon the success of LLM-as-a-judge (Zheng et al., 2024) and leverage AI feedback for self-improvement, an approach also known as RLAIF (Bai et al., 2022b; Lee et al., 2023). Yuan et al. (2024) and Wu et al. (2024) have further advanced this concept by employing LLMs to provide self-rewarding feedback on their own generated outputs. Unlike previous work that constrains the system from leveraging external signals, our framework allows for information inflow from external environments through the interactions of API calls, potentially elevating the upper bounds of self-improving.

3 SELF-IMPROVING VIA ACTIVE DATA SEARCH

The primary objective of our ACTIVE DATA SEARCH (ADS) framework is to endow AI systems with the capability to acquire valuable training data from external environments for retraining themselves, thereby facilitating self-improvement with minimal human supervision. As illustrated in Figure 1, we develop an *optimizer* model for a *policy* model. The optimizer model is designed to discover tailored training datasets to improve the policy by generating textual API calls to interact with various resources and environments. In this section, we start with a brief introduction to the problem formulation, followed by an overall description of policy optimization and optimizer refinement processes. We defer the details of our prototype implementations in Section 4.

3.1 PROBLEM FORMULATION

Given a target task T and a policy model $\mathcal{M}^p$, the goal of ADS is to maximize the performance of the policy model on T through active search for optimal training data. The data search process is conducted by an optimizer model $\mathcal{M}^o$ and a set of pre-defined APIs. Each API requires some input parameters and returns output data gathered from various environments. These APIs can be implemented through a variety of established methods, which may encompass but are not limited to data crawling, filtering, cleaning, refinement, synthesis, and manual annotation. The resultant data can also be sourced from a broad spectrum, such as internet search engines, LLM-based AI assistants, or crowdsourcing annotation services. To mitigate excessive and unnecessary API invocations, we assign varying costs to different APIs according to their practical complexities. Following common practice, each target task is represented by a set of instructions Q, which serve as input for the optimizer model. The output of this model is an action trajectory A, comprising multiple API calls.

3.2 Policy Optimization

The system optimizes its policy model $\mathcal{M}^p$ for a given target task T through the following steps. First, the optimizer model $\mathcal{M}^o$ is instructed to conduct a comprehensive analysis of the instructions Q, identifying the essential required capabilities. Following this, $\mathcal{M}^o$ performs an introspective assessment of its own proficiencies and deficiencies on these identified capabilities, and generates sequential API calls $A = [a_1, a_2, ..., a_K]$ for data acquisition, where K is the number of API calls and is determined on-the-fly. Notably, the above process is completed by the optimizer as free-form text generation, enabling seamless incorporation of any pre-defined APIs.

Next, we execute each API call a_k in the environment $\mathcal{E}$, and collect the returned data to obtain a training dataset D, which may include a variety of data types such as raw documents, supervised demonstrations, and preference data, following the best practice of current LLM training. Finally, we train the original $\mathcal{M}^p$ on the tailored training dataset to update its knowledge and capacities, resulting in an updated policy model $\hat{\mathcal{M}}^p$.

3.3 Optimizer Refinement

To make efficient data-collecting decisions, the optimizer model $\mathcal{M}^o$ should be aware of the strengths and weaknesses of the policy model $\mathcal{M}^p$. Therefore, we propose an iterative training process for optimizer refinement. First of all, we assume a set of training tasks $\mathcal{T} = \{T\}$, where reliable performance measurements are available. It is important to note that we *do not* make this assumption at test-time since the evaluation of arbitrary real-world tasks can be complex and resource-intensive. Instead, we posit that the self-knowledge developed for the optimizer during the training phase can be effectively generalized to novel test tasks.

We then employ reinforcement learning and consider both performance gains and API costs in reward design. The training is conducted iteratively. At each iteration, we sample multiple API trajectories $\mathcal{A} = \{A\}$ for each training task T. We then execute the data acquisition and policy training processes accordingly and collect the corresponding performance gain $S(A)$ and API cost $C(A)$. To avoid potential data leakage, we partition the instruction set Q of T into an observed set Q^o and another held-out set Q^h. The optimizer can only see the examples in the observed set and the performance gains are measured based on the held-out set. This results in a set of quadruple $\mathcal{B} = \{(T, A, S, C)\}$. We then optimize $\mathcal{M}^o_t$ to $\mathcal{M}^o_{t+1}$ on $\mathcal{B}$ with reinforcement learning algorithms such as rejection sampling (Bai et al., 2022a) and direct preference optimization (Rafailov et al., 2024). The complete process of optimizer refinement is described in Algorithm 1.

Algorithm 1 Optimizer Refinement Procedure in ADS

Require: Training tasks $\mathcal{T}$, policy model $\mathcal{M}^p$, optimizer model $\mathcal{M}^o$, environment $\mathcal{E}$.
for iteration t in N **do**
  Initialize optimizer buffer $\mathcal{B} = \{\}$.
  for task T in $\mathcal{T}$ **do**
    Split task instructions Q into observed set Q^o and held-out set Q^h.
    Sample API trajectories $\mathcal{A} \sim \mathcal{M}^o_t(\cdot|Q^o)$.
    for trajectory A in $\mathcal{A}$ **do**
      Execute A in $\mathcal{E}$ to acquire data to update $\mathcal{M}^p$ to $\hat{\mathcal{M}}^p$.
      Collect performance gain $S(A)$ of $\hat{\mathcal{M}}^p$ on Q^h, and API cost $C(A)$.
      Add (T, A, S, C) to $\mathcal{B}$.
    end for
  end for
  Optimize $\mathcal{M}^o_t$ to $\mathcal{M}^o_{t+1}$ using reinforcement learning on $\mathcal{B}$.
end for
return $\mathcal{M}^o_N$.

API Name	Parameter	Return	Description	Cost
`Information Retrieval`	Query	Document	Sparse and Dense Retrieval	Low
`Demonstration Generation`	Topic	Instruction-Response Pair	Synthesis with the Policy	Medium
`Question Answering`	Question	Question-Answer Pair	Annotated by Stronger LLM	High

Table 1: Implementation details of our data collecting APIs.

4 Experimental Setup

4.1 APIs

We design three distinct data-collecting APIs to facilitate the *acquisition*, *utilization*, and *enhancement* of knowledge. The first API is `Information Retrieval`, which employs both sparse and dense retrieval to retrieve relevant documents from external knowledge databases such as Wikipedia, thereby supporting knowledge acquisition, analogous to the pre-training stage of LLMs. The second API is `Demonstration Generation`, which utilizes the policy model to generate appropriate exemplar instruction-response pairs, tailored to various knowledge application scenarios, reminiscent of the alignment stages of LLMs. The last API is `Question Answering`, which resorts to the wisdom of human experts, mimicking how humans learn from each other. [1]. It provides to-the-point answers to questions proposed by the optimizer model. Notably, this framework is not restrictive and can accommodate additional APIs as needed. In each API trajectory, these APIs are invoked sequentially, e.g., `api_name_1(api_param_1)...api_name_n(api_param_n)`. Subsequently, the corresponding API calls are executed to formalize the training dataset. We then combine the training datasets from different API calls to update the policy model. For tasks where the model possesses the required capabilities, a "none" option is incorporated. We present the API implementation details in Table 1.

To avoid excessive API calls, e.g., asking questions that the policy model already knows the answer to, we take into account the API cost in our implementation. The cost associated with each API varies according to their practical complexities and computational overhead. Concretely, we assign relative costs as follows: `Information Retrieval` is at one since retrieval is fast and cheap, `Demonstration Generation` at two given generation is slow and resource-intensive, and `Question Answering` at three as it requires a more powerful model or manual efforts. Further implementation details of APIs are provided in Appendix A.

4.2 Dataset

To train and evaluate the optimizer model, it is essential to collect a diverse set of "target tasks" for which we aim to optimize the performance of the policy model. For this purpose, we adopt the general instruction-following dataset Llama-3-Magpie-Air-3M-v0.1 (Xu et al., 2024). Each instruction in this dataset is associated with several labels such as task category, intent, and difficulty. The instructions and labels are generated by Llama-3-8B-Instruct (Dubey et al., 2024). Subsequently, we group the instructions with identical labels into different clusters, with each cluster representing a distinct "target task". We discard tasks with fewer than five instructions. The resulting dataset comprises 10,239 distinct tasks, which we partition into 8,739 tasks for training, 500 tasks for validating, and 1,000 tasks for testing. In the train and valid splits, we allocate three observed instructions per task for trajectory generation and reserve two held-out instructions for performance measurement. This approach ensures that the optimizer learns to improve performance at *task level* rather than *instance level*, as over-fitting on the observed instructions may not necessarily translate performance gains on the held-out instructions. For the test split, to enhance the robustness of evaluation, we augment the initial five instructions per task to 100 using Self-Instruct (Wang et al., 2023), with three instructions as observed and 97 as held-out. More details of our dataset are shown in Appendix B.

4.3 Evaluation Methods

In-house Evaluation We employ both the RM and GPT-4 (Achiam et al., 2023) to compare responses generated by the original and updated policy models on held-out instructions across our test tasks. We report the average win, tie, and lose rate across all tasks. For RM evaluation, we sample five API trajectories and compute the average RM score among them, ensuring statistical robustness. For GPT-4 judgment, we employ a head-to-head comparison using the pairwise evaluation prompt

[1] For efficiency and reproducibility, we employ a stronger LLM as a proxy for human supervision in practice.

proposed by Zheng et al. (2024). We try two different response orders to prevent order sensitivity. To save token usage, we only sample one trajectory and test on two held-out instructions per task.

Generalizing to Public Benchmarks We additionally utilize three well-established benchmarks for further evaluating the generalizability of the trained optimizer models, including AlpacaEval 2.0 (Dubois et al., 2024), Arena-Hard (Li et al., 2024), and MT-Bench (Zheng et al., 2024). AlpacaEval 2.0 includes 805 instructions for daily chat scenarios while Arena-Hard contains 500 more challenging real-world questions. MT-Bench consists of 80 multi-turn dialogues spanning eight distinct domains[2]. Since the original version of MT-Bench contains incorrect reference responses, we follow previous works (Adler et al., 2024; Wan et al., 2024) to use the updated version for evaluation. The proposed approach aims to optimize the performance on specific target tasks. However, these benchmarks (AlpacaEval 2.0, Arena-Hard, and MT-Bench) cover a wide range of distinct tasks, which can be too broad to be considered as a single target task. To conduct meaningful experiments, we make the following adaptations. First, we cluster the original instructions based on task category and difficulty, utilizing the approach in Xu et al. (2024). This process yields 38, 34, and 21 "target tasks" for AlpacaEval 2.0, Arena-Hard, and MT-Bench, respectively. Then, we employ the Self-Instruct approach to generate three new instructions for each task and use them as the observed instructions for that task. We only assess model performance on the original instructions provided in these benchmarks. Following conventions (Meng et al., 2024; Adler et al., 2024), we conduct a standard evaluation using GPT-4 judgment. We adhere to the default setups and report the win rate of the updated policy models against the GPT-4 baseline for each benchmark. Additionally, we conduct a supplementary evaluation using RM as judgment and report the average adjusted win rate, calculated as win rate $+\ 0.5 \times$ tie rate. For both RM and GPT-4 judgments, we also report the weighted averages of these benchmarks based on their number of samples. Further details of the evaluation process can be found in Appendix D.

4.4 Training Details

We employ Qwen-2-7B-Instruct (Yang et al., 2024a) and Gemma-2-9B-Instruct (Riviere et al., 2024) as our policy models[3]. The optimizer model is initialized using the corresponding policy model.

During each iteration of ADS, the optimizer model generates five API trajectories per target task, based on the observed instructions. To further enhance the diversity of the trajectories, we incorporate three additional default trajectories, each corresponding to a specific API type. These trajectories are executed in the data-collecting environments to construct a comprehensive training dataset. Next, the policy model is updated through either in-context learning or fine-tuning using this training data. Considering the frequent policy model updating for each target task, we use in-context learning to maintain computational efficiency while preserving performance, since extensive research has demonstrated that in-context learning can achieve comparable effectiveness to fine-tuning (Mosbach et al., 2023; Agarwal et al.). We then determine the reward signal of each API call trajectory by evaluating the performance gains of the updated policy model on the held-out instructions. We use FsfairX-Llama-3-RM-v0.1 (Xiong et al., 2024) as the RM for evaluation. Finally, we iteratively update the optimizer model by direct preference optimization (DPO) (Rafailov et al., 2024) on the chosen and rejected trajectories with the highest and lowest rewards. To ensure training stability, we perform warm-up rejection sampling (RS) (Bai et al., 2022a) on the chosen trajectories exhibiting the highest rewards before iterative DPO for optimizer model training. In Section 6.1, we further illustrate the comparison between DPO and RS, as well as investigate the impact of the default API trajectories. The iterative training process of the optimizer model is presented as follows:

- Prompting: $\mathcal{M}^o$ is initialized from $\mathcal{M}^p$ without fine-tuning, then direct prompting.
- RS Iteration 0: $\mathcal{M}^o_0$ is initialized from $\mathcal{M}^o$, then warm-up RS on chosen API trajectories.
- DPO Iteration 1, 2, 3: $\mathcal{M}^o_1$, $\mathcal{M}^o_2$, $\mathcal{M}^o_3$ are initialized from $\mathcal{M}^o_0$, $\mathcal{M}^o_1$, $\mathcal{M}^o_2$, then DPO on chosen and rejected API trajectories.

To enhance the optimizer model's capacity for reward improvements while minimizing API costs, we introduce a novel cost-control approach, which draws inspiration from the length-control strategy

[2]Given that our dataset focuses on single-turn tasks, we evaluate the 1st turn performance for consistency.

[3]Since our training data are generated by Llama-3-8B-Instruct, we exclude it from our choices of policy models to avoid any potential biases.

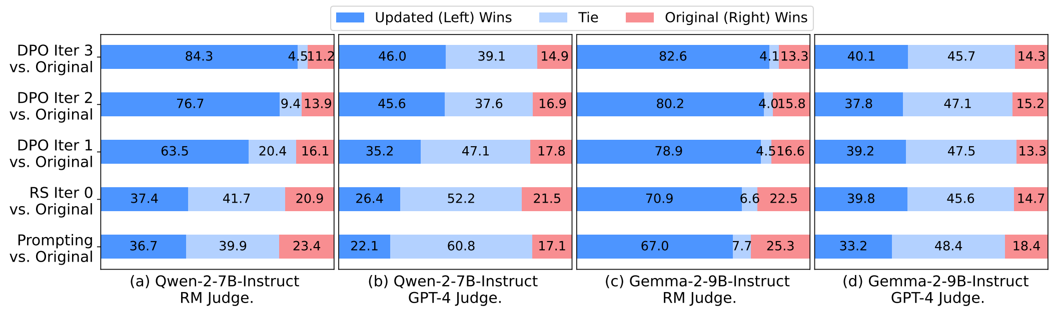

Figure 2: Comparisons between responses generated by the updated and original policy models across in-house test tasks with both RM and GPT-4 judgments. The updated policy models are developed on training data collected by different iterations of optimizer models.

employed in LLM alignment (Wu et al., 2024). Our approach posits that among trajectories with comparable rewards, those associated with lower costs are more valuable for optimization and thus deserve higher reinforcement feedback. Specifically, we introduce a cost tier parameter $\tau \in [0, 1]$ to control the trade-off between rewards and costs. Trajectories within the top-tier rewards ranging $[(1-\tau)R_{\max} + \tau R_{\min}, R_{\max}]$ are considered to have similar performance. From this subset, we select the trajectory with the lowest cost as the chosen trajectory. Conversely, for the reject trajectory, we select the one with the highest cost within $[R_{\min}, (1-\tau)R_{\min} + \tau R_{\max}]$. To maintain an optimal balance between reward and cost, we empirically determine the cost tier parameter at 0.1. We further conduct experiments to illustrate the effectiveness of the proposed cost-control approach in Section 6.2. More information on training details can be found in Appendix C.

5 Main Results

5.1 In-house Evaluation

Figure 2 illustrates the evaluation results on target tasks across in-house test split. We compare the responses generated by the updated and original policy models, displaying the task win, tie, and lose rates with both RM and GPT-4 judgments.

ADS presents significant superiority across 1,000 in-house test tasks. We can first observe that the prompting method yields slight performance gains. For instance, the updated policy model achieves a win rate of 36.7% compared to the original policy model's 23.4% in RM judgment, and 22.1% to 17.1% in GPT-4 judgment for Qwen-2-7B-Instruct. Following the iterative training process, the final model exhibits a remarkable enhancement, with a win rate of 84.3% versus 11.2% in RM evaluation and 46.0% versus 14.9% in GPT-4 evaluation. These substantial performance improvements can be attributed to the valuable training data discovered by the optimizer model in ADS, which enhances the knowledge and capabilities required to accomplish these test tasks.

Iterative ADS boosts consistent performance improvements. During the initial iteration of DPO training, our optimizer model exhibits rapid adaptation to high-reward API trajectories. Consequently, the updated policy model achieves substantial improvements in win rates: from 37.4% to 63.5% according to RM judgment, and from 26.4% to 35.2% as evaluated by GPT-4 for Qwen-2-7B-Instruct. As the number of training iterations increases, the updated policy model consistently demonstrates improved win rates. Given that the training tasks remain constant across different iterations, we posit that the consistent performance gains are from the automatic weakness identification and refinement during the iterative training process, thus progressively increasing the probabilities of generating optimal API trajectories for self-improvement.

5.2 Public Benchmarks

In Table 2, we present the evaluation results on three public benchmarks, including AlpacaEval 2.0, Arena-Hard, and MT-Bench. We compare the responses generated by the policy model and the GPT-4 baseline model and show the win rates with both RM and GPT-4 judgments.

Models	RM Judgment AE	AH	MT	Avg.	GPT-4 Judgment AE	AH	MT	Avg.
Qwen-2-7B-Instruct ($\mathcal{M}^p$)	31.7	56.1	66.9	42.6	24.0	25.6	55.9	26.4
Prompting ($\mathcal{M}^o \rightarrow \mathcal{M}^p$)	33.2	57.0	68.1	43.8	24.9	26.7	52.5	27.2
RS Iteration 0 ($\mathcal{M}^o_0 \rightarrow \mathcal{M}^p$)	32.4	56.7	68.1	43.2	24.2	26.7	53.8	26.8
DPO Iteration 1 ($\mathcal{M}^o_1 \rightarrow \mathcal{M}^p$)	34.9	57.9	66.9	45.0	28.6	28.0	53.8	29.8
DPO Iteration 2 ($\mathcal{M}^o_2 \rightarrow \mathcal{M}^p$)	36.5	60.6	69.4	47.1	30.6	28.8	57.7	31.5
DPO Iteration 3 ($\mathcal{M}^o_3 \rightarrow \mathcal{M}^p$)	**38.8**	**61.1**	**71.9**	**48.8**	**31.9**	**30.1**	**59.4**	**32.8**
Δ *to Qwen-2-7B-Instruct*	(+22.3%)	(+8.9%)	(+7.5%)	(+14.6%)	(+32.9%)	(+17.7%)	(+6.2%)	(+24.3%)
Gemma-2-9B-Instruct ($\mathcal{M}^p$)	28.6	**70.0**	**71.3**	46.0	34.8	**37.5**	55.0	36.9
Prompting ($\mathcal{M}^o \rightarrow \mathcal{M}^p$)	28.0	59.2	70.0	41.7	33.8	30.0	56.3	33.7
RS Iteration 0 ($\mathcal{M}^o_0 \rightarrow \mathcal{M}^p$)	30.5	65.6	**71.3**	45.5	35.3	34.8	51.3	36.0
DPO Iteration 1 ($\mathcal{M}^o_1 \rightarrow \mathcal{M}^p$)	30.4	48.4	66.9	39.0	36.7	22.4	54.4	32.6
DPO Iteration 2 ($\mathcal{M}^o_2 \rightarrow \mathcal{M}^p$)	**33.0**	67.9	70.6	**47.8**	36.2	35.2	**57.5**	37.1
DPO Iteration 3 ($\mathcal{M}^o_3 \rightarrow \mathcal{M}^p$)	32.8	65.7	69.4	46.8	**37.0**	37.4	56.9	**38.3**
Δ *to Gemma-2-9B-Instruct*	(+14.5%)	(-6.1%)	(-2.6%)	(+1.6%)	(+6.3%)	(-0.2%)	(+3.4%)	(+3.7%)
Llama-3.1-70B-Instruct	53.0	76.7	83.8	63.3	39.5	57.0	75.0	47.9
Qwen-2-72B-Instruct	38.9	66.8	73.1	50.9	35.1	48.3	66.9	41.7
Gemma-2-27B-Instruct	33.6	80.2	75.0	52.8	37.0	49.9	60.6	43.0

Table 2: Comparisons between responses generated by the updated policy models and the baseline GPT-4 on AlpacaEval 2.0 (AE), Arena-Hard (AH), and MT-Bench (MT) with both reward model (RM) and GPT-4 judgment. Percentages indicate the relative improvements/decreases observed in the final iteration of the updated policy model when compared to the original policy model.

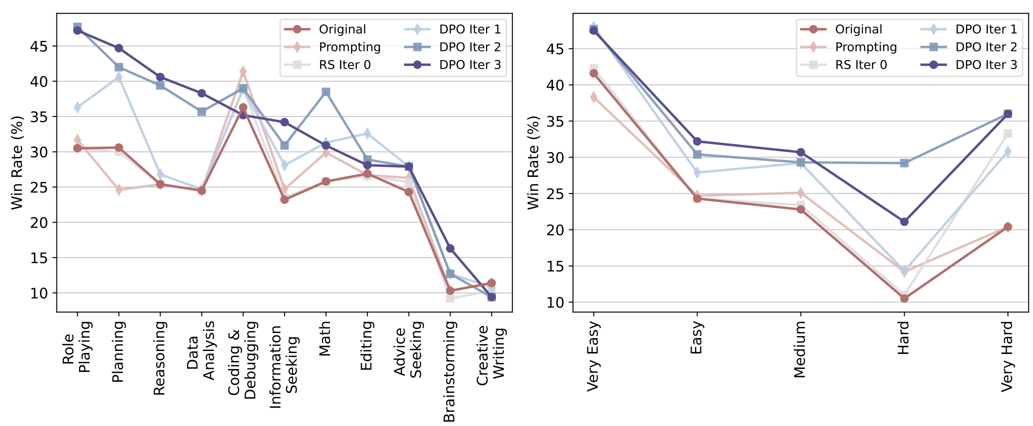

Figure 3: Fine-grained evaluation of different updated policy models and the original Qwen-2-7B-Instruct across various task categories (left) and difficulties (right) in AlpacaEval 2.0.

ADS demonstrates generalized capabilities on public benchmarks. For the target tasks from public benchmarks, our approach exhibits generalized improvements. Overall, the updated policy model of Qwen-2-7B-Instruct shows significant relative gains across all benchmarks, with an average enhancement of +14.6% and +24.3% in RM and GPT-4 judgment, respectively. Notably, in AlpacaEval 2.0, the improvements are even more pronounced, e.g., +22.3% in RM evaluation and +32.9% in GPT-4 evaluation. These findings reveal that our optimizer model, despite being trained on limited target tasks, demonstrates the capacity to find valuable training data for practical tasks, effectively addressing the challenges for real-world task optimization.

ADS enhances policy model to rival that of more powerful LLMs. We show that with the help of training data collected from our optimizer model, smaller and weaker policy models can achieve comparable results to those of larger and stronger LLMs. Specifically, in the RM evaluation within AlpacaEval 2.0, Qwen-2-7B-Instruct with ADS achieves a win rate of 38.8%, which is almost equivalent to the performance of the substantially larger Qwen-2-72B-Instruct. Similarly, in the GPT-4 evaluation, Gemma-2-9B-Instruct with ADS, achieves a win rate of 37.0%, matching the performance of larger Gemma-2-27B-Instruct.

ADS typically enhances knowledge-intensive and reasoning-related tasks. We conduct a fine-grained evaluation across distinct task categories in AlpacaEval 2.0. The results of Qwen-2-7B-

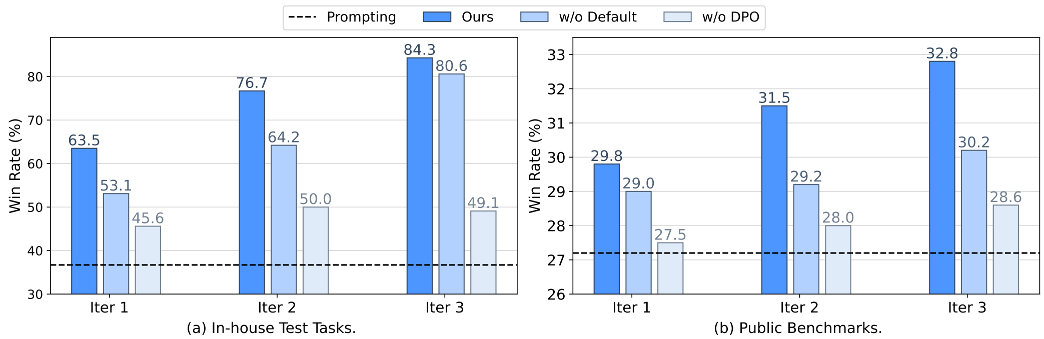

Figure 4: Ablation study results of Qwen-2-7B-Instruct across in-house test tasks with RM judgment (left) and public benchmarks with GPT-4 judgment (right), where "w/o DPO" refers to the replacement of direct preference optimization with rejection sampling algorithm, while "w/o Default" denotes the exclusion of default API trajectories for optimizer model training.

Instruct are illustrated in Figure 3 (left). We find that compared to the original baseline, ADS improves the performance in most of the categories, especially for those that require substantial general or specific knowledge and advanced reasoning abilities, such as information-seeking, role-playing, reasoning, and planning. However, in categories like editing, creative writing, and coding & debugging, our approach only has slight improvements or maintains comparable to the baseline. This limited enhancement can be potentially attributed to the inherent nature of these tasks, which primarily involve format and style rewriting, as well as fragment modifications, presenting inherent challenges for optimization through in-context learning from acquired training data.

ADS particularly improves in complex and challenging tasks. In Figure 3 (right), we further demonstrate the performance across tasks with various levels of difficulty. The results reveal that ADS yields more substantial improvements as task complexity increases. Specifically, the relative performance enhancement increases from +14.2% for very easy tasks to a remarkable +76.5% for tasks categorized as very hard. These results further demonstrate the potential of ADS for continuous self-improving, typically in complex tasks that lack expert supervision.

6 Ablation and Analysis

6.1 Ablation Studies

In this section, we delve into the key components in our practical implementation of ADS, focusing on the training algorithm and the construction of API trajectories. We first investigate and compare the performance of the optimizer model when trained iteratively using rejection sampling and direct preference optimization. Subsequently, we analyze the influence of adding default API trajectories into the optimizer model's training process. The evaluation results across in-house test tasks with RM judgment and public benchmarks with GPT-4 judgment are shown in Figure 4.

DPO enhances discrimination between chosen and rejected trajectories. We can observe that compared to rejection sampling, the implementation of direct preference optimization substantially improves the optimizer model's capacity to differentiate between chosen and rejected API trajectories, which is illustrated by a significant increase in the average win rate from 49.1% to 84.3% across in-house test tasks. Similarly, in public benchmarks, the win rate improved from 28.6% to 32.8%. This enhanced discriminative capability facilitates more effective weakness identification and decision-making processes of our optimizer model, ultimately leading to high-reward API trajectory exploration and optimal training data acquiring for deficiency enhancement.

Incorporating default API trajectories enhances trajectory diversity. In addition to the five trajectories sampled by the optimizer during its training process, we incorporate three default trajectories, each corresponding to a distinct data-collecting API in our implementation. Table 4 in Appendix C illustrates an example of both default and generated trajectories. The inclusion of these default API trajectories alongside the self-generated candidates results in a more diverse set of trajectories for optimizer model training, facilitating improved average win rates across all iterations, both in in-house test tasks and public benchmarks.

6.2 Analysis Results

We further make an in-depth analysis to investigate the factors contributing to the efficacy of ADS, focusing on two key aspects: the advantages of self-explored training data and the implementation of the cost-control mechanism. To evaluate the impact of self-explored training data, we compare the API trajectories generated by our optimizer model against those curated using a baseline strategy. This baseline approach utilizes the `Question Answering` API for each observed instruction in the target task to construct the corresponding API trajectory. Subsequently, we employ the collected data for policy model optimization in both cases. Regarding the cost-control mechanism, we compare our method with an alternative approach that prioritizes reward maximization without considering API costs. Figure 5 illustrates these analytical results, encompassing both in-house test tasks using RM judgment (left) and public benchmarks with GPT-4 judgment (right).

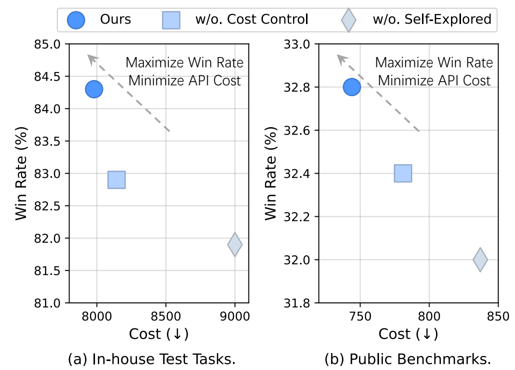

Figure 5: Effectiveness of self-explored training data and cost-control mechanism in maximizing win rate while minimizing API cost.

Self-explored data presents more suitable for LLMs training. Despite employing the most sophisticated and expensive `Question Answering` API for training data construction, the approach without self-explored training data achieves a relatively lower win rate, e.g., 81.9% versus 84.3% across in-house test tasks, while incurring higher API costs, e.g., 9,000 versus 8,219, compared to our approach. The optimizer model in ADS is designed to automatically identify and address the potential capability gaps in specified tasks based on the developed self-knowledge. Consequently, our approach provides a more tailored and targeted performance improvement.

Cost-control mechanism reduces the API cost while improving performance. To maximize the potential for self-improvement while maintaining appropriate resource allocation, we implement a cost-control mechanism that optimizes the trade-off between minimizing costs and maximizing performance during the training process of the optimizer model, as detailed in Section 4.4. In comparison to the approach that focuses solely on maximizing performance without considering costs, our method not only achieves a lower cost as expected but also demonstrates an improved win rate. The performance improvements can be attributed to the increased trajectory diversity compared to the reward maximization approach. This observation indicates that our approach enables the development of a robust and cost-effective system for data acquisition, ultimately contributing to improved overall performance and reduced computational overhead.

7 Conclusion & Limitations

In this study, we explored enabling LLMs to autonomously acquire optimal training data for self-improvement with minimal human intervention. We proposed a novel framework, Active Data Search (ADS), which utilizes LLMs themselves as an optimizer to strategically invoke appropriate APIs, facilitating the discovery of tailored training datasets from external environments for self-training. To ensure efficient data-collecting decisions, we introduced an iterative refinement algorithm for the optimizer, guided by reinforcement feedback signals aiming to maximize task performance while minimizing computation costs. Through a series of experiments on 1,000 in-house test tasks and three public benchmarks, we demonstrated the effectiveness and generalizability of ADS. Notably, ADS exhibited the capacity to enhance the performance of smaller and weaker language models to be comparable with larger and stronger LLMs on AlpacaEval 2.0. This automated process of data discovery and self-training presents opportunities to reduce the reliance on human expertise in LLM development, providing a new direction for future research in this domain.

The current implantation is a proof-of-concept with several limitations. First, the optimizer and policy models are separate. Unifying them is an interesting avenue for future work. Second, the APIs included currently are far from covering all existing data techniques. Third, we did not consider multi-turn optimization of the policy models in the framework.

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*, abs/2303.08774, 2023.

Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, et al. Nemotron-4 340b technical report. *ArXiv preprint*, abs/2406.11704, 2024.

Rishabh Agarwal, Avi Singh, Lei M Zhang, Bernd Bohnet, Luis Rosias, Stephanie CY Chan, Biao Zhang, Aleksandra Faust, and Hugo Larochelle. Many-shot in-context learning. In *ICML 2024 Workshop on In-Context Learning*.

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *ArXiv preprint*, abs/2312.11805, 2023.

Beijing Academy of Artificial Intelligence BAAI. Infinity instruct, 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*, abs/2204.05862, 2022a.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, et al. Constitutional ai: Harmlessness from ai feedback. *ArXiv preprint*, abs/2212.08073, 2022b.

Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, Thomas Wolf, and Leandro von Werra. Cosmopedia: how to create large-scale synthetic data for pre-training, 2024.

Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, et al. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. In *Forty-first International Conference on Machine Learning*, 2024.

Jiale Cheng, Yida Lu, Xiaotao Gu, Pei Ke, Xiao Liu, Yuxiao Dong, Hongning Wang, Jie Tang, and Minlie Huang. Autodetect: Towards a unified framework for automated weakness detection in large language models. *ArXiv preprint*, abs/2406.16714, 2024.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, SHUM KaShun, and Tong Zhang. Raft: Reward ranked finetuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *ArXiv preprint*, abs/2407.21783, 2024.

Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy S Liang, and Tatsunori B Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (rest) for language modeling. *ArXiv preprint*, abs/2308.08998, 2023.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *ArXiv preprint*, abs/2306.11644, 2023.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2024.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, Heather Miller, et al. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, et al. Rlaif: Scaling reinforcement learning from human feedback with ai feedback. *ArXiv preprint*, abs/2309.00267, 2023.

Nicholas Lee, Thanakul Wattanawong, Sehoon Kim, Karttikeya Mangalam, Sheng Shen, Gopala Anumanchipali, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Llm2llm: Boosting llms with novel iterative data enhancement. *ArXiv preprint*, abs/2403.15042, 2024.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *ArXiv preprint*, abs/2406.11939, 2024.

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. Textbooks are all you need ii: phi-1.5 technical report. *ArXiv preprint*, abs/2309.05463, 2023.

Fei Liu, Tong Xialiang, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Forty-first International Conference on Machine Learning*, 2024a.

Tianqi Liu, Yao Zhao, Rishabh Joshi, Misha Khalman, Mohammad Saleh, Peter J Liu, and Jialu Liu. Statistical rejection sampling improves preference optimization. In *The Twelfth International Conference on Learning Representations*, 2024b.

Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. On llms-driven synthetic data generation, curation, and evaluation: A survey. In *Findings of the Association for Computational Linguistics*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations*, 2019.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu, 2024.

Chris Lu, Samuel Holt, Claudio Fanconi, Alex J Chan, Jakob Foerster, Mihaela van der Schaar, and Robert Tjarko Lange. Discovering preference optimization algorithms with and for large language models. *ArXiv preprint*, abs/2406.08414, 2024.

Yu Meng, Mengzhou Xia, and Danqi Chen. Simpo: Simple preference optimization with a reference-free reward. *ArXiv preprint*, abs/2405.14734, 2024.

Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 12284–12314, 2023.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4. *ArXiv preprint*, abs/2306.02707, 2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35, 2022.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *ArXiv preprint*, abs/2408.00118, 2024.

Corby Rosset, Ching-An Cheng, Arindam Mitra, Michael Santacroce, Ahmed Awadallah, and Tengyang Xie. Direct nash optimization: Teaching language models to self-improve with general preferences. *ArXiv preprint*, abs/2404.03715, 2024.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2023.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in Neural Information Processing Systems*, 33, 2020.

Zhiqing Sun, Yikang Shen, Qinhong Zhou, Hongxin Zhang, Zhenfang Chen, David Cox, Yiming Yang, and Chuang Gan. Principle-driven self-alignment of language models from scratch with minimal human supervision. *Advances in Neural Information Processing Systems*, 36, 2024.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. Stanford alpaca: An instruction-following llama model, 2023.

Fanqi Wan, Ziyi Yang, Longguang Zhong, Xiaojun Quan, Xinting Huang, and Wei Bi. Fusechat: Knowledge fusion of chat models. *ArXiv preprint*, abs/2402.16107, 2024.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *The Twelfth International Conference on Learning Representations*, 2024.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023.

Leandrovon Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, and Shengyi Huang. Trl: Transformer reinforcement learning, 2020.

Alexander Wettig, Aatmik Gupta, Saumya Malik, and Danqi Chen. Qurating: Selecting high-quality data for training language models. In *Forty-first International Conference on Machine Learning*, 2024.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020.

Tianhao Wu, Weizhe Yuan, Olga Golovneva, Jing Xu, Yuandong Tian, Jiantao Jiao, Jason Weston, and Sainbayar Sukhbaatar. Meta-rewarding language models: Self-improving alignment with llm-as-a-meta-judge. *ArXiv preprint*, abs/2407.19594, 2024.

Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighof. C-pack: Packaged resources to advance general chinese embedding. *ArXiv preprint*, abs/2309.07597, 2023.

Tim Z Xiao, Robert Bamler, Bernhard Schölkopf, and Weiyang Liu. Verbalized machine learning: Revisiting machine learning with language models. In *ICML 2024 Workshop on In-Context Learning*, 2024.

Wei Xiong, Hanze Dong, Chenlu Ye, Ziqi Wang, Han Zhong, Heng Ji, Nan Jiang, and Tong Zhang. Iterative preference learning from human feedback: Bridging theory and practice for rlhf under kl-constraint. In *Forty-first International Conference on Machine Learning*, 2024.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *ArXiv preprint*, abs/2304.12244, 2023.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. *ArXiv preprint*, abs/2406.08464, 2024.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *ArXiv preprint*, abs/2407.10671, 2024a.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024b.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E Weston. Self-rewarding language models. In *Forty-first International Conference on Machine Learning*, 2024.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *ArXiv preprint*, abs/2406.07496, 2024.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35, 2022.

Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. Offline training of language model agents with functions as learnable weights. In *Forty-first International Conference on Machine Learning*, 2024.

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

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36, 2024.

Fan Zhou, Zengzhi Wang, Qian Liu, Junlong Li, and Pengfei Liu. Programming every example: Lifting pre-training data quality like experts at scale. *ArXiv preprint*, abs/2409.17115, 2024a.

Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, et al. Symbolic learning enables self-evolving agents. *ArXiv preprint*, abs/2406.18532, 2024b.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*, 2023.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jurgen Schmidhuber. Language agents as optimizable graphs. *ArXiv preprint*, abs/2402.16823, 2024.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *ArXiv preprint*, abs/1909.08593, 2019.

A Details of APIs

In this section, we first present the prompt for API trajectory generation of the optimizer model in Figure 6, then show the detailed implementation of our data collecting APIs, including (1) `Information Retrieval`, which facilitates efficient knowledge acquisition; (2) `Demonstration Generation`, which enables various knowledge utilization scenarios; and (3) `Question Answering`, which serves to enhance and refine the acquired knowledge.

Prompt for API Trajectory Generation

Task Overview
Your goal is to analyze input prompts, identify knowledge gaps, and strategically use provided APIs to enhance your knowledge and capabilities.

Provided APIs
You will have access to the following APIs to obtain additional data for improvements:
1. "information_retrieval(query: string)": Retrieves relevant documents for a given search query. Use for factual knowledge gaps.
2. "example_instantiation(topic: string)": Generates practical instances based on a given topic. Use for applying knowledge to concrete situations.
3. "question_answering(question: string)": Provides answers to a given question. Use for deeper understanding of knowledge.

Constraints and Guidelines
1. Focus on common and general knowledge and capabilities requirements across all the input prompts.
2. Use API calls only to address competence gaps when necessary, not to directly solve the prompts.
3. If you need to make API calls, formatted as <api_calls><api>api_name_1(api_param_1) </api><api>api_name_2(api_param_2) </api>...</api_calls>
4. If you do not need to make API calls, formated as <api_calls>none</api_calls>

Input:
{observed_instructions}

Figure 6: Prompt for API trajectory generation of the optimizer model. {observed_instructions} is the placeholder for the observed instructions in the target task.

Information Retrieval For our retrieval corpus, we utilize the Wikipedia (en) from December 20th, 2022 [4], encompassing approximately 8.59 million paragraphs. Our retrieval process employs a two-stage approach. Initially, we implement sparse retrieval using the BM25 [5] algorithm to identify the top 1,000 most relevant documents for a given search query. Subsequently, we refine this selection through dense retrieval, leveraging the BGE-Large-EN-v1.5 (Xiao et al., 2023) embedding model to obtain the most appropriate document from the previously identified candidates.

Demonstration Generation To obtain knowledge utilization examples, we employ an approach that relies solely on the policy model itself without incorporating external tools. Specifically, we leverage the policy model to synthesize demonstrations by generating instructions and corresponding responses based on a given knowledge topic. The prompt used for demonstration generation is shown in Figure 7.

Question Answering To enhance the comprehension of acquired knowledge, we employ Llama-3.1-70B-Instruct (Dubey et al., 2024) as a replacement for human experts to generate comprehensive responses to complex questions. The prompt used for question answering is illustrated in Figure 8.

[4] `https://huggingface.co/datasets/Cohere/wikipedia-22-12`
[5] `https://github.com/xhluca/bm25s`

Prompt for Demonstration Generation

Generate an Instruction and the corresponding comprehensive Response related to the Topic.

Topic: {topic}

Figure 7: Prompt for Demonstration Generation. {topic} is the placeholder for the knowledge topic.

Prompt for Question Answering

Provide a detailed Answer to the given Question.

Question: {question}

Figure 8: Prompt for question answering. {question} is the placeholder for the given question.

B Details of Dataset

We group instructions into clusters based on three key attributes: category, intent, and difficulty. The category attribute represents the broad task type, encompassing areas such as creative writing, reasoning, and coding. The intent attribute indicates the primary objectives within the instructions, including getting helpful tips, identifying logical fallacy, develop software extensions. The difficulty attribute quantifies the complexity of following the instructions, ranging from very easy to very hard.

To improve the robustness of evaluation results, we augment the size of the held-out set for each target task in the test split. Specifically, we leverage five existing instructions as seed examples, and prompt GPT-4 to generate new instructions that are significantly different from these initial examples but belong to the same task (Wang et al., 2023). To maintain distinctiveness, we employ a filtering mechanism whereby any generated instructions with a Rouge-L similarity score exceeding 0.7 when compared to the original instructions are eliminated. The prompt for instruction augmentation is illustrated in Figure 9. The statistics of our train, valid, and test splits are shown in Table 3.

Statistics	Train	Valid	Test
# Task	8,739	500	1,000
# Category	12	12	12
# Intent	4,982	371	902
# Difficulty	5	5	5
# Obs. Inst. Per Task	3	3	3
# Held. Inst. Per Task	2	2	97

Table 3: Statistics of train, valid, and test splits.

C Details of Training Process

For policy model updating, we employ ICL on the collected training dataset for each target task, including retrieved documents, instantialized instruction-response pairs, and answers to complex questions, as the supplementary information to address the given held-out instruction. The prompt for ICL of the policy model is shown in Figure 10.

For optimizer model updating, We train the optimizer model with a batch size of 128 and a maximum sequence length of 2048. The training is conducted on a single node with 8x80GB Nvidia A100 GPUs for one epoch per iteration. We perform RS for a single iteration and DPO for three iterations. The models are optimized using the AdamW (Loshchilov & Hutter, 2019) optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We use a weight decay of 0.0 and gradient clipping of 1.0. A cosine learning rate schedule is employed, with a warmup ratio of 0.1 and a maximum learning rate of 2e-5 for RS and 5e-7 for DPO. The β parameter in DPO is set to 0.01. Our training framework is developed based on the HuggingFace Transformers (Wolf et al., 2020) and TRL (Werra et al., 2020). We show an example of default and generated API trajectories in Table 4.

Prompt for Task-Specific Instructions Augmentation

Given 5 instructions as demonstrations within a specific task, please generate 100 new distinct instructions that are relevant to the same task but significantly different from the demonstrations provided.

Task: {task}

Here are 5 demonstrations:
Instruction 1: {seed_instruction_1}
Instruction 2: {seed_instruction_2}
Instruction 3: {seed_instruction_3}
Instruction 4: {seed_instruction_4}
Instruction 5: {seed_instruction_5}

Now, please generate 100 new distinct instructions that are:

1. Significantly different from the demonstrations.
2. Within the same task or domain as the demonstrations.
3. Similar in complexity and difficulty to the demonstrations.
4. Varied in their approach to the task (e.g., different aspects or sub-tasks within the task).
5. Consistent with the tone and style of the demonstrations.

Please format your generated instructions as follows:

Generated Instruction 1: [Your generated instruction]
Generated Instruction 2: [Your generated instruction]
Generated Instruction 3: [Your generated instruction]
...

Figure 9: Prompt for task-specific instructions augmentation. {task} is the placeholder for the given task. {seed_instruction_i} is the placeholder for i-th seed instruction.

Prompt for ICL

{collected_data}

Your task is to read the above text to answer the following question.

1. Note that the above information is only visible to you, don't indicate that you have read above text in your answer. For example, don't say "according to the information provided above".
2. Do not solely rely on the provided information. You know much more than the provided information. Use it as a hint to write an comprehensive, informative, elaborate, and helpful answer.

{held_out_instruction}

Figure 10: Prompt for ICL of the policy model. {collected_data} is the placeholder for the training dataset derived from the optimizer model. {held_out_instruction} is the placeholder for the held-out instruction in the target task.

Example of Default and Generated API Trajectories

Observed Instructions:
Explain the implications of the Einstein-Podolsky-Rosen paradox on quantum entanglement.
Describe the mathematical formulation of the wave function collapse in the Copenhagen interpretation.
Analyze the potential applications of quantum tunneling in developing next-generation transistors.

Generated API Trajectory:
Analysis of Requirements

After analyzing the input prompts, I have identified the required background knowledge and broader capabilities as follows:
1. Advanced knowledge of quantum physics
2. Ability to explain complex scientific concepts
3. Understanding of quantum mechanics principles
4. Familiarity with practical applications of quantum theory

Competence Assessment

I have knowledge gaps in quantum mechanics principles and capability gaps in explaining complex scientific concepts. Therefore, I need to make API calls for this set of prompts.

API Calls

<api_calls>
<api>retrieve_information("Quantum mechanics principles")</api>
<api>get_instances("Quantum theory application")</api>
<api>ask_question("How to describe and analysis complex scientific concepts?")</api>
</api_calls>

Default API Trajectory:
Analysis of Requirements

After analyzing the input prompts, I have identified the required background knowledge and broader capabilities.

Competence Assessment

I have knowledge gaps and capability gaps. Therefore, I need to make API calls for this set of prompts.

API Calls

<api_calls>
<api>ask_question("Explain the implications of the Einstein-Podolsky-Rosen paradox on quantum entanglement.")</api>
<api>ask_question("Describe the mathematical formulation of the wave function collapse in the Copenhagen interpretation.")</api>
<api>ask_question("Analyze the potential applications of quantum tunneling in developing next-generation transistors.")</api>
</api_calls>

Table 4: An example of default and generated API trajectories for a set of observed instructions.

D Details of Evaluation Setups

We first show the detailed implementation of our in-house evaluation and public benchmarks in Table 5. Then, we provide the corresponding task statistics of AlpacaEval 2.0, Arena-Hard, and MT-Bench in Table 6 and Table 7 by category and difficulty, respectively.

Benchmark	# Task	# Ins.	Baseline Model	RM Judgment	GPT-4 Judgment
In-house Test Split	1,000	97,000	Original Policy Model	FsfairX-Llama-3-RM-v0.1	GPT-4-1106-Preview
AlpacaEval 2.0	38	805	GPT-4-1106-Preview	FsfairX-Llama-3-RM-v0.1	GPT-4-1106-Preview
Arena-Hard	34	500	GPT-4-0314	FsfairX-Llama-3-RM-v0.1	GPT-4-1106-Preview
MT-Bench	21	80	GPT-4-0314	FsfairX-Llama-3-RM-v0.1	GPT-4-0125-Preview

Table 5: Implementation details of evaluation on in-house test tasks and public benchmarks.

Category	AlpacaEval 2.0		Arena-Hard		MT-Bench	
	Number	Percentage	Number	Percentage	Number	Percentage
Advice Seeking	79	9.8%	12	2.4%	4	5.0%
Brainstorming	25	3.1%	5	1.0%	1	1.3%
Coding & Debugging	42	5.2%	204	40.8%	7	8.8%
Creative Writing	54	6.7%	6	1.2%	5	6.3%
Data Analysis	7	0.9%	66	13.2%	3	3.8%
Editing	73	9.1%	19	3.8%	5	6.3%
Information Seeking	381	47.3%	63	12.6%	21	26.3%
Math	40	5.0%	56	11.2%	17	21.3%
Planning	61	7.6%	45	9.0%	3	3.8%
Reasoning	27	3.4%	17	3.4%	11	13.8%
Role Playing	14	1.7%	4	0.8%	3	3.8%

Table 6: The task category statistics of AlpacaEval 2.0, Arena-Hard, and MT-Bench.

Difficulty	AlpacaEval 2.0		Arena-Hard		MT-Bench	
	Number	Percentage	Number	Percentage	Number	Percentage
Very Easy	27	3.4%	0	0.0%	1	1.3%
Easy	491	61.0%	81	16.2%	25	31.3%
Medium	256	31.8%	339	67.8%	52	65.0%
Hard	25	3.1%	79	15.8%	2	2.5%
Very Hard	6	0.7%	1	0.2%	0	0.0%

Table 7: The task difficulty statistics of AlpacaEval 2.0, Arena-Hard, and MT-Bench.

E Comparison between ADS and Data Construction Baselines

In this section, we expand our experimental evaluation by comparing the proposed ADS framework with several baseline methods for training data construction. These methods include: (1) **Prompting**, which constructs API trajectories through optimizer model prompting without fine-tuning; (2) **Retrieval Augmentation**, which employs both sparse and dense retrieval to identify relevant documents based on target task instructions, similar to the `Information Retrieval` API; (3) **Self-Instruct**, which utilizes the policy model to generate new instruction-response pairs for the target task, functioning analogously to the `Demonstration Generation` API; (4) **Rule-based QA**, which leverages an stronger LLM to answer each instruction in the target task, comparable to the `Question Answering` API. The empirical results presented in Table 8 indicate that ADS, incorporating all three APIs, significantly outperforms these baseline methods across both in-house test tasks and public benchmarks, and maintaining its simplification without human intervention.

Methods	Qwen-2-7B-Instruct		Gemma-2-9B-Instruct	
	In-house Test Tasks	Public Benchmarks	In-house Test Tasks	Public Benchmarks
Prompting	36.7	27.2	67.0	33.7
Retrieval Augmentation	24.2	26.8	46.4	32.6
Self-Instruct	55.8	31.6	76.9	35.1
Rule-based QA	81.9	32.0	79.7	36.0
ADS	**84.3**	**32.8**	**82.6**	**38.3**

Table 8: Comparison between ADS and training data construction baseline methods.

F INFLUENCE OF THE INSTRUCTION-FOLLOWING DATASET

Since we adopt the instructions from Llama-3-Magpie-Air-3M-v0.1 (Xu et al., 2024) as the task instructions in our optimizer training process, we conduct a controlled experiment to ensure a fair comparison. Specifically, we fine-tune our base models using this dataset and evaluate their performance on public benchmarks before and after fine-tuning. As shown in Table 9, the fine-tuning process leads to a notable degradation in model performance. This decline can be attributed to the fact that our base models (Qwen-2-7B-instruct and Gemma-2-9-Instruct) have already undergone extensive fine-tuning with high-quality training data, whereas the Llama-3-Magpie-Air-3M-v0.1 dataset may contain relatively lower-quality data. These empirical findings substantiate that the performance improvements observed in our experiments stem from the effectiveness of the proposed ADS, rather than from the instruction-following dataset.

Methods	Qwen-2-7B-Instruct				Gemma-2-9B-Instruct			
	AlpacaEval 2.0	Arena-Hard	MT-Bench	Average	AlpacaEval 2.0	Arena-Hard	MT-Bench	Average
Base Model	**24.0**	**25.6**	**55.9**	**26.4**	**34.8**	**37.5**	**55.0**	**36.9**
Base Model w/ Magpie	11.9	11.7	43.1	13.6	14.8	12.0	44.9	15.5

Table 9: Comparison between the base models before and after fine-tuning on the instruction-following dataset, where "w/ Magpie" denotes the base models after fine-tuning.