Divide, Optimize, Merge: Scalable Fine-Grained Generative Optimization for LLM Agents

Jiale Liu[1], **Yifan Zeng**[2], **Shaokun Zhang**[1], **Chi Zhang**[3], **Malte Højmark-Bertelsen**[5], **Marie Normann Gadeberg**[5], **Huazheng Wang**[2,4], **Qingyun Wu**[1,4]
[1]Pennsylvania State University [2]Oregon State University
[3]The University of Texas at Austin [4]AG2AI, Inc. [5]Beyond Work
{jiale.liu, shaokun.zhang, qingyun.wu}@psu.edu
{zengyif, huazheng.wang}@oregonstate.edu
chizhang@cs.utexas.edu
{malte, marie}@beyondwork.ai

Abstract

LLM-based optimization has shown remarkable potential in improving agentic systems. However, the conventional approach of prompting LLM-based generative optimizer with the trajectories on the whole training dataset in a single pass becomes untenable as datasets grow, leading to context window overflow and degraded pattern recognition. To address these challenges, we propose Fine-grained Generative Optimization (FGO), a scalable framework that divides large optimization tasks into manageable subsets, performs targeted optimizations, and systematically combines optimized components through progressive merging. Evaluation across ALFWorld, LogisticsQA, and GAIA benchmarks demonstrates that FGO outperforms conventional approach by 1.6-8.6% while reducing average prompt token consumption by 56.3%. Our framework provides a practical solution for scaling up LLM-based generative optimization of increasingly sophisticated agentic systems. Further analysis demonstrates that FGO achieves the most consistent performance gain in all training dataset sizes, showcasing its scalability and efficiency.

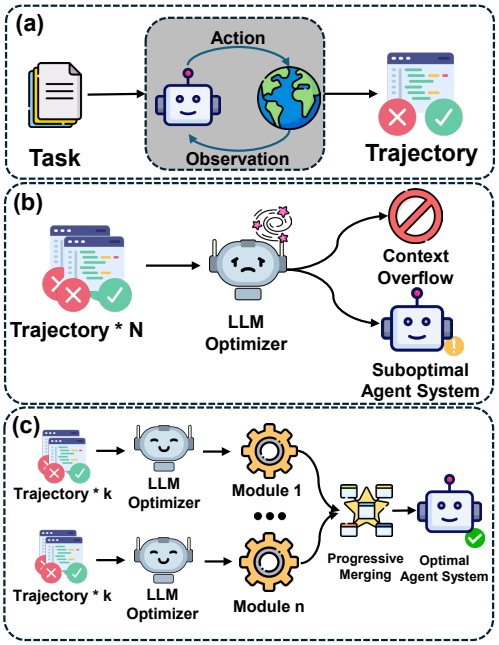

Figure 1: Overview of agent optimization. (a) Basic agent execution process. (b) Traditional all-at-once optimization faces context overflow and deteriorated performance. (c) Our method: divide-and-conquer with progressive merging enables scalable optimization.

1 Introduction

Large Language Models (LLMs) have emerged as powerful optimizers for agentic systems, capable of analyzing execution trajectories and refining system modules like prompts (Yang et al., 2024; Zhou et al., 2022; Khattab et al., 2023; Opsahl-Ong et al., 2024; Pryzant et al., 2023), tools (Qian et al., 2023; Zhang et al., 2024c,b; Wang et al., 2024a; Yuan et al., 2023). This paradigm, termed generative optimization (Yang et al., 2024; Cheng et al., 2024), leverages the LLM's inherent ability to comprehend and learn from execution patterns, allowing for systematic improvements without requiring access to model weights or gradients. The optimized agentic system has shown great potential in various domains, including reasoning (Cheng et al., 2024; Zelikman et al., 2023), software engineering (Jimenez et al., 2023; Pan et al., 2024; Zelikman et al., 2023), data analysis (Hu et al., 2024b; Jing et al., 2024), computer operation (Wang et al., 2025; Xie et al., 2025; Abuelsaad et al., 2024; Xia and Luo, 2025).

However, due to the increasing volume of data required for optimizing LLM agentic systems autonomously, directly applying LLM-based optimization approaches encounters a fundamental scalability issue. Existing methods typically concatenate all execution trajectories on the training data and perform optimization in an all-at-once manner, feeding the entire dataset into the LLM optimizer in a single prompt. While this approach works for optimization tasks with small-scale data,

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 18990–19012

it becomes problematic as the data grows. For instance, in the GAIA benchmark (Mialon et al., 2023), agents are tasked to retrieve and analyze information from a diverse range of sources, including web and different types of files. Solving a task from the dataset requires agents to interact with external tools or forge their own ones in order to collect real-world information and perform reasoning. The sophisticated process can generate lengthy execution traces for subsequent optimization, which is filled with raw documents and complex intermediate reasoning steps, even challenging for human to parse (Zhang et al., 2025). This increasing complexity leads to two critical limitations: (1) The concatenated trajectories exceed LLM context windows, forcing truncation of valuable optimization signals. (2) Even when content fits within context windows, LLMs struggle with analyzing long-range dependencies in extensive corpus (Bai et al., 2024; Liu et al., 2024; Ni et al., 2024; Ravaut et al., 2024; Li et al., 2024), making it hard for the LLM optimizer to capture subtle patterns and relationships between execution traces. As a result, such approaches can produce suboptimal solutions, particularly in complex scenarios where understanding the intricate relationships between different execution trajectories is crucial for improving agent performance.

To address these scalability challenges, we introduce Fine-grained Generative Optimization (FGO), a framework that enables efficient optimization of LLM-based agentic systems with large-scale data. Specifically, FGO operates through three components: (1) Task division that breaks down the large training dataset into more manageable subsets, (2) Fine-grained generative optimization enabling efficient processing of each subset, and (3) Progressive module merging that adaptively combines the individually optimized modules while preserving crucial insights from each subset. This design allows FGO to effectively handle larger optimization tasks while maintaining high-quality results.

We evaluate FGO by optimizing two agent modules: instruction prompts and tools an agent could access, on diverse tasks including ALFWorld (Shridhar et al., 2020), LogisticsQA, and GAIA (Mialon et al., 2023). Agents trained with FGO produce significant performance gains across all datasets, ranging from 8.3% to 38.1%, outperforming other optimization methods by 1.6%-8.6%. Further analysis reveals that FGO maintains superior performance across varying training dataset sizes, highlighting its scalability and stability. Notably, FGO achieves these improvements while reducing prompt token consumption by 56.3% and increasing optimization efficiency by 7.6% compared to conventional all-at-once optimization.

Our contributions are threefold: (1) We identify and analyze the scalability limitations in current LLM-based optimization approaches for agentic systems. (2) To address the scalability limitation, we propose FGO, a scalable optimization framework that effectively handles large-scale agent optimization through task division and progressive merging. (3) We conduct extensive experiments across diverse tasks to validate FGO's effectiveness and provide insights into its scalability advantages through comprehensive empirical analysis.

2 Preliminary

2.1 Problem Setup

LLM Agent Optimizable Modules Agentic systems exhibit complex behavioral patterns emerging from multiple factors. A critical insight in designing such systems lies in the decomposition of the agent's parameter space into *modules* that can be independently optimized (Anthropic, 2024a). This decomposition enables targeted optimization of specific functional aspects while maintaining global system coherence. Denote the parameter space of agentic system as Θ, which partitions into trainable modules $\{\Theta_i\}_{i=1}^n$ governing distinct behavioral dimensions. Each module must satisfy two key properties to qualify as a modular unit. First, the *trainability* property requires that each module can influence the agent's policy gradients when exposed to queries. This ensures the module is sufficiently responsive to reward signals during optimization. Second, the *orthogonality* property requires that parameter gradients across different modules exhibit minimal directional alignment during optimization. Such orthogonality constraint ensures modules encode non-redundant functionalities while guaranteeing each contributes uniquely to performance optimization.

Agentic System Optimization An agent interacts with an environment $\mathcal{E}$ by generating a sequence of actions in response to a query q. Given parameters $\theta \in \Theta$, the agent's policy π determines actions a_t based on the current state of interaction history $a_{1:t-1}$ and observations $o_{1:t}$. These actions, combined with the observations from the environment,

form a trajectory τ that represents the agent's solution attempt for the given query. To put it formally:

$$a_t \sim \pi(\cdot|a_{1:t}, o_{1:t}; \theta),\ o_{t+1} \sim \mathcal{E}(\cdot|a_t),\ \forall t \in [T]$$
$$\tau = \mathcal{A}(q; \theta) = (o_1, a_1, ..., o_T, a_T) \quad (1)$$

The performance is quantified through a loss function $\mathcal{L}$. Given a distribution $\mathcal{D}$ over query-label pairs (q, y), we aim to find optimal agent parameters that minimize expected loss across the task distribution. The optimization objective is:

$$\theta^* = \arg\min_{\theta \in \Theta} \mathbb{E}_{(q,y)\sim\mathcal{D}} \left[\mathcal{L}(\mathcal{A}(q; \theta), y)\right] \quad (2)$$

This formulation of optimization via tuning modules provides a unified abstraction for analyzing performance-critical factors in agentic system design. In practice, the modules include prompts for task handling (Wen et al., 2024; Wu et al., 2024b), long term memory (Zhang et al., 2024d), the available toolbox (Zhang et al., 2024c), and the weights of the backbone LLM (Zeng et al., 2024; Ma et al., 2024).

2.2 Motivation

In generative optimization setting, we assume the numeric value of the policy gradient is not accessible in Eq. 2. This constraint emerges from a practical reality in modern LLM agent systems - the increasing reliance on proprietary LLMs like GPT-4 (OpenAI, 2023) and Claude (Anthropic, 2024b), where internal parameters are inaccessible.

Current approaches that leverage LLM as optimizer typically follow a two-step iterative process: first evaluating modules on training data to collect trajectories and losses, then prompting the LLM optimizer with this information to generate improved modules. While these methods have shown promising results (Yang et al., 2024; Zhang et al., 2024c; Cheng et al., 2024), they face fundamental scalability challenges that limit their practical applications.

Context window limit. The inherent constraint of context window is a critical bottleneck in generative optimization. As the number of training samples grows, concatenated trajectories often exceed the context capacity of even the most capable LLMs. This limitation becomes particularly acute in complex tasks with extensive reasoning steps or multi-turn interactions. Even a modest number of samples can overwhelm the context window, severely restricting the optimizer's ability to process comprehensive training data.

Insufficient context utilization. Even when content fits within context limits, LLMs struggle to effectively process patterns across extensive trajectory collections (Ni et al., 2024; Li et al., 2024; Bai et al., 2024). Recent benchmarks consistently show LLM performance deteriorates with increasing text length, particularly for complex dialogues and documents (Bai et al., 2024; Wu et al., 2024a; Ni et al., 2024). For LLM-based optimizers, this limitation is critical as optimization requires understanding long-range dependencies and fine-grained details across multiple samples. Consequently, LLM-based optimizers often produce suboptimal module updates that fail to capture the full complexity of real-world optimization problems where both broad patterns and specific details matter.

3 Methods

3.1 Overview

The overall pipeline of FGO is illustrated in Figure 2. The core concept behind our proposed framework is to divide the large task set into smaller, more manageable subsets and optimize them independently. After we obtain the optimal modules trained on each subsets, we develop an algorithm to progressively merge them into an optimal module.

3.2 Fine-Grained LLM Agent Optimization

Basic Module Optimization We first describe our approach to agent optimization, with the process illustrated in Algorithm 1. Each optimization epoch consists of three phases: exploration, evaluation, and optimization. During exploration, the agent uses its current module parameters to interact with given tasks, generating solution trajectories. In the evaluation phase, an LLM-based evaluator analyzes these trajectories to determine correctness, identify failure points, recognize patterns, and highlight improvement areas based on ground truth and execution details. These evaluations serve as textual gradients that guide module updates toward improved performance. Finally, in the optimization phase, an LLM-based optimizer processes the trajectories and evaluations to synthesize an updated module with enhanced performance characteristics.

Divide As the number and complexity of task set scales, the length and number of the trajectories can quickly increase, posing challenge to LLM-based optimization. To address the issue, we propose a divide-and-conquer based strategy that decomposes the training dataset D into N disjoint

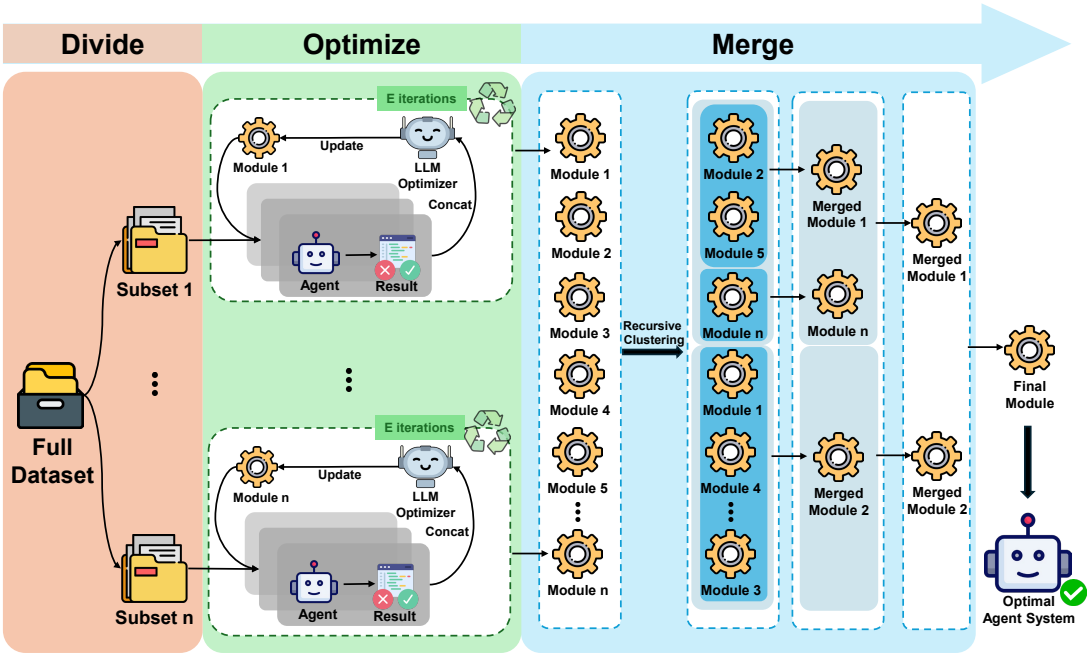

Figure 2: Illustration of FGO's optimization pipeline. The system operates in three stages: (1) Divide: the full dataset is split into manageable subsets, (2) Optimize: parallel optimization is performed on each subset using LLM-based optimization with multiple iterations, and (3) Merge: optimized modules are progressively combined using recursive clustering to produce the final optimal agent system.

Algorithm 1 Module Optimization

Input: Task set $\mathcal{D}$, number of epochs E
Output: Optimized module θ
$\theta \leftarrow \phi$ ▷ Start from scratch
for $e \leftarrow 1$ to E **do**
  $\mathcal{H} \leftarrow \{\}$ ▷ Empty trajectory history
  for $(q, y) \in \mathcal{D}$ **do**
    $\tau \leftarrow \mathcal{A}(q; \theta)$ ▷ Eq. 1
    $r \leftarrow \text{Evaluate}(\tau, y)$
    $\mathcal{H}.\text{append}((\tau, r))$
  end for
  $\theta \leftarrow \text{LLM}_{\textbf{optim}}(\mathcal{H}, \theta)$ ▷ Update module
end for
return θ

Algorithm 2 Progressive Module Merging

Input: List $\mathcal{M} = \{(\theta_i, \mathcal{T}_i, p_i)\}$ containing modules, tasks, and performances, cluster size threshold t
Output: Optimized module θ^*, performance p^*
function PROGRESSIVEMERGE($\mathcal{M}$, t)
  if $|\mathcal{M}| \leq t$ **then**
    $\theta, p \leftarrow \text{Merge}(\mathcal{M})$ ▷ Base: Direct Merge
    return θ, p
  end if
  $C \leftarrow \text{KMeans}(S, \lfloor\sqrt{|\mathcal{M}|}\rfloor)$ ▷ Adaptive cluster
  for each cluster $c_i \in C$ **do**
    $\theta_i, p_i \leftarrow \text{ProgressiveMerge}(c_i, t)$
  end for
  return $\text{Merge}(\{\theta_i, p_i\}_{i=1}^{|C|})$
end function
return ProgressiveMerge($\mathcal{M}$, t)

subsets $\{D_i\}_{i=1}^{N}$, and perform optimization on the subsets independently. By operating on smaller, focused subsets, the intuition is to capture subtle patterns and requirements that might be overlooked in global optimization. The process yields N distinct module-loss pairs, each specialized for its respective subset's characteristics.

Progressive Merging While decomposition addresses immediate scalability constraints, it introduces the challenge of effectively integrating N independently optimized modules while preserving their specialized insights. A straightforward approach would be to directly prompt an LLM with all module-performance pairs to generate a unified module. However, such all-at-once merging often fails to effectively process and synthesize patterns across numerous modules simultaneously, potentially losing the specialized optimizations gained through divided optimization. We propose progressive merging, implemented as a recursive al-

gorithm that strategically controls merging granularity through a cluster size threshold. Algorithm 2 details the progressive merging process. For a given list of module-performance pairs, we first check if the list size exceeds the predefined threshold t. For larger lists, we partition the modules into $k = \lfloor \sqrt{n} \rfloor$ clusters based on their similarities[1], where n is the number of modules. Each resulting cluster then undergoes recursive merging. When a cluster's size falls below the threshold t, we merge its constituent modules by prompting an LLM with their contents and corresponding performance statistics. After each merge operation, we evaluate the merged module's performance by validating it on the combined task set from all constituent modules. This recursive process naturally constructs a bottom-up merging tree, where each internal node represents a validated merge of its children's modules. This controlled, progressive approach ensures that each merge operation remains within LLM context limits while capturing intricate relationships between similar modules, ultimately enabling efficient optimization of agentic systems.

4 Evaluations

4.1 Experiment Setup

We evaluate FGO by optimizing two distinct and critical modules of agentic system: prompts and tools. Effective prompts serve as essential guidelines that enable agents to properly comprehend task requirements and execution constraints, significantly enhancing task completion capabilities (Fu et al., 2024; Chen et al., 2024; Wu et al., 2024b; Zhao et al., 2024). Meanwhile, tools represent specialized functional components that expand the agent's action space, providing targeted capabilities for solving domain-specific challenges.

Datasets We conduct experiments on three different benchmarks to study the performance of FGO.

- **ALFWorld** (Shridhar et al., 2020) is a text-based benchmark for household task planning, where agents navigate virtual environments through natural language commands. The agent receives a high-level objective and must execute a sequence of appropriate actions to accomplish tasks. The environment provides immediate success/failure feedback upon task completion. We evaluate performance using success rate across various task categories as well as the overall average success rate across all tasks.
- **LogisticsQA** is our own curated benchmark. The dataset consists of UBL format shipping invoice documents from real world scenarios. The agent is tasked to understand and extract the transport reference number from the document. A task is considered successful if the agent's extracted answer is an exact match with the ground truth. Please refer to Appendix E for details of dataset.
- **GAIA** (Mialon et al., 2023) is a benchmark designed to test the capability of agents as general assistants. It encompasses tasks from different domains such as file browsing, web searching and scraping, making it a perfect testbed for benchmarking agent's tool usage capability as well as the quality of the toolbox. The benchmark presents significant challenges in tool selection, information integration, and reasoning across multiple sources of information. A task is considered successful if the agent's answer matches exactly with the ground truth.

Baselines for Comparison. We include ProTeGi (Pryzant et al., 2023), Reflexion (Shinn et al., 2024), MIPROv2 (Opsahl-Ong et al., 2024) for comparing instruction optimization, and CRAFT (Yuan et al., 2023), AgentOptimizer (Zhang et al., 2024c) for comparing tool optimization. We also include the algorithm 1 and its variants to compare the module optimization outcome. (1) **All-at-once** represents performing agent optimization on the whole training set using the alg. 1; (2) **Batch-wise** employs a fixed-size batching strategy, splitting the training dataset into predetermined chunks and performing optimization the batches sequentially; (3) **Bootstrapping** implements a stochastic approach, sampling task batches from the training dataset with replacement. For a full detailed description of the baselines and the implementation details, please refer to Appendix B.

4.2 Main Results

Finding 1: FGO demonstrates superior optimization performance across multiple domains. As shown in Table 1, FGO consistently outperforms both established baselines and alternative implementations across multiple domains. In the prompt optimization task, FGO achieves the highest average score on ALFWorld, surpassing the strongest baseline MIPROv2 by 6.7%. While MIPROv2 exhibits competitive performance on in-

[1]The modules studied are represented as text. We calculate cosine similarities based on their embeddings.

Methods	ALFWorld							LogisticsQA
	Pick	Clean	Heat	Cool	Look	Pick 2	Avg.	
Base Agent	69.4	50.5	65.2	20.6	31.5	21.6	45.5	36.3
ProTeGi	66.7	54.8	60.9	81.0	55.6	23.5	58.2	44.7
Reflexion	70.8	60.2	82.6	85.7	72.2	64.7	72.2	49.8
MIPROv2	**95.8**	67.7	78.3	81.0	77.8	47.1	75.4	61.2
All-at-once	90.2	72.6	78.3	78.6	66.7	55.9	75.0	52.1*
Batch-wise	77.1	71.0	67.4	64.3	**86.1**	**73.5**	72.8	55.7
Bootstrapping	91.7	77.4	73.9	74.6	87.0	41.2	75.6	62.6
FGO	90.2	**83.8**	**87.0**	**88.9**	**86.1**	62.7	**83.6**	**64.8**

Table 1: Performance of the optimized agent using different optimization methods on ALFWorld and LogisticsQA. The best results are in bold. * denotes that we encounter context window exceeded error during optimization and have to trim the context sent to the LLM optimizer.

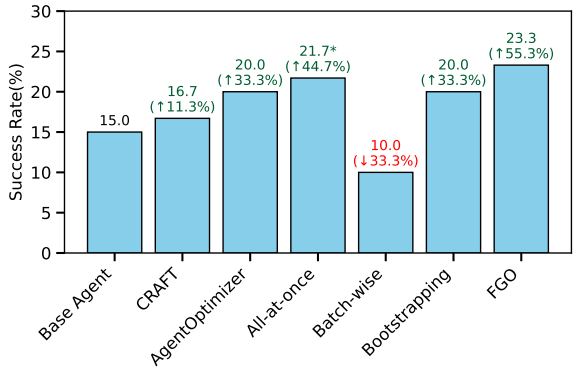

Figure 3: Performance of the optimized agent using different optimization methods on GAIA. * denotes that we encounter context window exceeded error during optimization and have to trim the context sent to the LLM optimizer.

dividual subtasks, our method excels by decomposing the dataset and applying targeted optimizations to each subset. This task type-specific approach enables FGO to address the unique challenges in different environments more effectively than general-purpose optimizers. For tool optimization, FGO demonstrates a clear advantage on GAIA, outperforming other methods. While approaches like AgentOptimizer incorporate fail-safe mechanisms for stability, the generated tools miss corner cases. In contrast, FGO's merging mechanism examines different optimization outcomes across subsets to generate a holistic toolset covering edge cases. Overall, FGO achieves the most substantial performance improvements across all evaluation scenarios, with enhancements ranging from 8.3% to 38.1% compared to the vanilla base agents, demonstrating the effectiveness of our fine-grained optimization approach.

Finding 2: Progressive merging effectively preserves task-specific patterns while achieving global optimum. FGO's superior performance stems from its divide-and-conquer methodology. The All-at-once approach struggles with processing complex patterns across the entire dataset, resulting in suboptimal performance on specific ALFWorld subtasks. While bootstrapping and batch-wise optimization show strong results in certain categories, they lack consistency across the task spectrum. In contrast, FGO first optimizes subset-specific prompts and tools, then progressively merges them to preserve task-specific patterns while building toward global optimum. We examine the effects of merging on FGO performance in Section 4.4.

4.3 Further Analysis

Finding 3: FGO demonstrates extraordinary scalability. We evaluate how the volume of training data affects the optimization performance in ALFWorld[2]. As shown in Figure 4, FGO maintains stable performance across all dataset sizes, with consistent improvement as number of training samples increases. MIPROv2 also maintains stable performance due to its advanced algorithms, but achieves lower overall performance due to lack of targeted optimization. While batch-wise optimization shows similar training accuracy in low-data settings, it yields lower performance compared to bootstrapping optimization, indicating poorer generalization. This aligns with established machine

[2]Since Reflexion operates on a task basis and directly optimizes the test performance, the notion of training data does not apply. Therefore we do not include Reflexion in the plot.

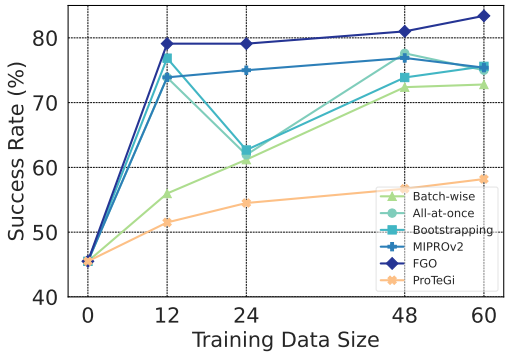

Figure 4: Analysis of the number of training tasks. We run optimization on varied training dataset sizes and plot the performance. FGO achieves best performance in all training settings, and demonstrate a steady increase.

learning principles where bootstrapping enhances generalization (Breiman, 1996). Additionally, All-at-once optimization proves impractical for LogisticsQA and GAIA due to the extensive solution trajectories exceeding context limit, which justifies the need for more scalable approach.

Finding 4: FGO achieves an optimal balance between token cost, efficiency and performance. We visualize the relationship between token consumption during optimization and the optimized performance in Figure 5, and outline the time to train and performance in Figure 7. In terms of token consumption, FGO requires larger number of tokens compared to Batch-wise and Bootstrapping optimization. This is because the merging process requires evaluating on the combined task set from all the constituent modules. This is a sacrifice in exchange for accurately modeling the merged module's capability in order to generate more accurate modules in the merging process. In terms of efficiency, FGO can perform optimization in parallel and gather the optimized modules at once, which is an unique advantage compared to the sequential training methods.

4.4 Ablation Study

We investigate the following questions to understand the impact of different design choice in FGO:

How do progressive merging and choice of clustering algorithm affect performance? To analyze the impact of progressive merging and clustering algorithms during merging, we conduct ablation experiments on ALFWorld. We first establish a baseline by removing progressive merging entirely, simply prompting an LLM to merge all module-

Cluster Algorithm	ALFWorld Avg of 3	ALFWorld Best of 3
None	73.1 (+27.6)	84.3 (+22.4)
Spectral	81.6 (+36.1)	89.6 (+27.7)
Bisect K-Means	80.1 (+34.6)	**91.0 (+29.1)**
K-Means	**83.6 (+38.1)**	89.6 (+27.7)

Table 2: Ablation study on the effects of clustering algorithms. "None" means we skip the clustering step and directly merge the optimized modules. The numbers in parentheses indicate the performance improvement over the base agent.

performance pairs directly. We then evaluate different clustering methods including Spectral clustering and Bisect K-Means. Results in Table 2 indicate that even when progressive merging is not applied, the method achieves an average success rate of 73.1%, representing a significant improvement over the base agent. Furthermore, incorporating progressive merging enhances performance across all clustering algorithms, suggesting that the progressive merging process itself plays a more critical role in boosting performance than the choice of clustering algorithm.

How does different partitioning scenario affect FGO's performance? To examine the robustness of FGO, we inspect two partitioning scenarios: (1) imbalanced partitioning, where subset sizes are highly skewed (controlled by parameter β, with smaller values indicating more imbalance); (2) random partitioning, where training tasks are randomly assigned to different partitions. Details about imbalanced dataset generation are in Appendix B. As shown in Table 3, an unbalanced dataset partition leads to deteriorated performance. Noticeably, even with highly skewed distributions, the performance drop is only 10% compared to category-based partitioning, still outperforming most baselines. This indicates that the adaptive merging process effectively captures underlying patterns despite suboptimal partitioning. Overall, FGO demonstrates resilience to suboptimal partitioning scenarios, maintaining competitive performance even in challenging conditions.

How does FGO perform with advanced reasoning LLM as optimizer backbone? To test FGO with advanced reasoning LLM as optimizer backbone, we use o3-mini to optimize the agents on AFLWorld and measure the resulting metrics. We set the reasoning effort to high for all cases. As shown in Table 4, FGO maintains the best perfor-

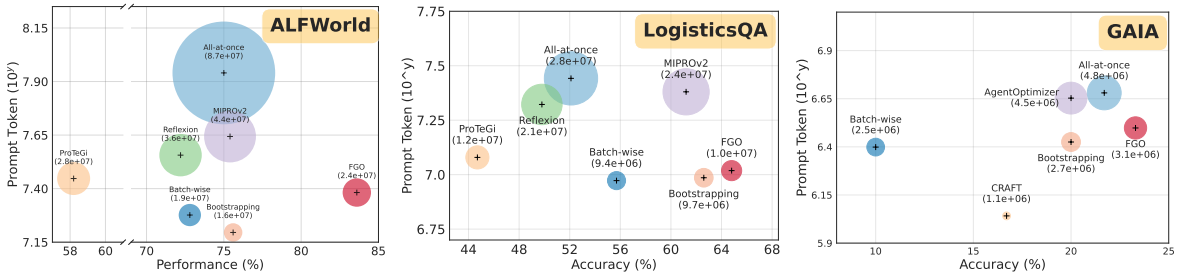

Figure 5: Comparison of prompt token efficiency across different optimization methods on ALFWorld, LogisticsQA, and GAIA. Each panel plots the trained agent's performance against the total prompt tokens consumed during optimization. Circle diameters are proportional to the optimization token consumption, with crosses (+) indicating circle centers.

Scenario	ALFWorld	
	Avg of 3	Best of 3
Imb. ($\beta = 0.6$)	73.4	85.1
Imb. ($\beta = 0.8$)	76.9	87.3
Random	80.3	88.1
Category	83.6	**89.6**

Table 3: Ablation on data partitioning scenario. In imbalanced setting, the distribution of the partitions are skewed and controlled by a parameter β. The smaller β, the more imbalanced the partitioning. In random setting, the training tasks are randomly assigned to different partitions instead of according to their categories.

Method	# Tokens	Time (s)	Avg of 3	Best of 3
All-at-once	8.15×10^7	7583	87.8	93.3
Batch-wise	1.59×10^7	2969	83.1	92.3
Bootstrapping	$\mathbf{1.34}\times10^7$	2521	88.8	95.0
FGO	1.97×10^7	**2142**	**89.3**	**95.5**

Table 4: Ablation on the optimizer backbone. We leverage reasoning model o3-mini as the backbone for optimization, and report the metrics. Best result is in bold.

mance while still achieving the highest efficiency in training time, with only a modest increase in token consumption compared to baselines.

How does the number of divided subsets affect performance? To answer this question, we conduct ablation study on the number of independent agent optimizers. We train agents on LogisticsQA and set the number of divided subsets to 3, 4, 6, 8, 12, respectively. Due to the high cost in running `gpt-4o`, we downsample 100 tasks from the test set and validate the optimized agent's performance. We plot the relationship between performance and training time in Figure 6. The results show that while accuracy remains mostly stable within a narrow range, the training time varies significantly. For example, using 8 subsets achieves the highest

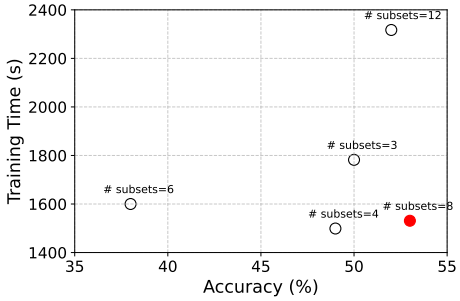

Figure 6: Ablation study on the number of independent agent optimizers. Most parameter settings achieve similar performance, with varying time for optimization.

accuracy (∼53%) in around 1525 seconds, whereas using 12 subsets requires over 2300 seconds for a slightly lower accuracy. This suggests that the choice of the number of subsets mostly impacts computational efficiency rather than the final optimization quality.

5 Related Work

LLM as Optimizer. LLMs are increasingly used as a blackbox optimizer for different LLM systems. In prompt optimization, LLM is leveraged to automously maximizing LLM's performance to novel tasks without expensive model tuning (Zhou et al., 2022; Pryzant et al., 2023; Cheng et al., 2023; Prasad et al., 2022; Khattab et al., 2024). In the realm of in-context learning (Min et al., 2021; Dong et al., 2022; Brown, 2020), by automatically retrieving demonstrations from training set (Zhao et al., 2021; Lu et al., 2021; Liu et al., 2021) or from adaptively annotated samples by LLM (Zhang et al., 2023; Wu et al., 2022; Su et al., 2022), prompt with autonomously selected in-context examples can reach performance better can human crafted prompts. LLM-based optimizers are also used as

meta-optimizers to debug and improve an LLM-based agentic system (Zelikman et al., 2023; Yin et al., 2024; Zhang et al., 2025; Song et al., 2024).

Automated Agentic System Design. Efforts in enhancing LLM inference performance have evolved since their emergence (Shinn et al., 2024; Madaan et al., 2024; Yao et al., 2023, 2024; Wei et al., 2022). This optimization paradigm has naturally extended to agentic systems, with recent works optimizing agent workflows through complex graphs (Zhuge et al., 2024), code (Hu et al., 2024a), and trees (Zhang et al., 2024a). Others focus on developing reusable tools (Zhang et al., 2024c; Cai et al., 2023; Qian et al., 2023; Yuan et al., 2023) and using past experiences (Zhao et al., 2024; Wang et al., 2024b) to improve performance.

6 Conclusion

In this paper, we address the scalability challenges in LLM-based agent optimization by introducing FGO, a framework that optimizes system modules through task division, fine-grained targeted optimization, and progressive merging. Evaluation across multiple benchmarks demonstrates that FGO consistently outperforms existing optimization approaches. Analysis shows that FGO achieves an optimal balance between performance and cost. FGO's scalability makes it valuable for optimizing sophisticated agentic systems in real-world cases.

Limitations

The merging process introduces computational overhead, as it requires back-testing the merged module on the merged training dataset, resulting in larger token cost compared to Batch-wise optimization and Bootstrappingoptimization. In future work, we attempt to leverage LLM to predict the performance of the merged module using in-context learning, or approximate the performance using Bayesian methods.

References

Tamer Abuelsaad, Deepak Akkil, Prasenjit Dey, Ashish Jagmohan, Aditya Vempaty, and Ravi Kokku. 2024. Agent-e: From autonomous web navigation to foundational design principles in agentic systems. *arXiv preprint arXiv:2407.13032*.

Anthropic. 2024a. Building effective agents. https://www.anthropic.com/research/building-effective-agents.

Anthropic. 2024b. Model card addendum: Claude 3.5 haiku and upgraded claude 3.5 sonnet.

Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2024. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. *arXiv preprint arXiv:2412.15204*.

Leo Breiman. 1996. Bagging predictors. *Machine learning*, 24:123–140.

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

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2023. Large language models as tool makers. *arXiv preprint arXiv:2305.17126*.

Minghao Chen, Yihang Li, Yanting Yang, Shiyu Yu, Binbin Lin, and Xiaofei He. 2024. Automanual: Generating instruction manuals by llm agents via interactive environmental learning. *arXiv preprint arXiv:2405.16247*.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. 2024. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms. *Advances in Neural Information Processing Systems*, 37:71596–71642.

Jiale Cheng, Xiao Liu, Kehan Zheng, Pei Ke, Hongning Wang, Yuxiao Dong, Jie Tang, and Minlie Huang. 2023. Black-box prompt optimization: Aligning large language models without model training. *arXiv preprint arXiv:2311.04155*.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, and Zhifang Sui. 2022. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of context-aware guidelines for large language model agents. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Shengran Hu, Cong Lu, and Jeff Clune. 2024a. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*.

Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, et al. 2024b. Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. 2024. Dsbench: How far are data science agents to becoming data science experts? *arXiv preprint arXiv:2409.07703*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*.

Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. 2024. Long-context llms struggle with long in-context learning. *arXiv preprint arXiv:2404.02060*.

Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2021. What makes good in-context examples for gpt-3? *arXiv preprint arXiv:2101.06804*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2021. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. *arXiv preprint arXiv:2104.08786*.

Zixian Ma, Jianguo Zhang, Zhiwei Liu, Jieyu Zhang, Juntao Tan, Manli Shu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Caiming Xiong, et al. 2024. Taco: Learning multi-modal action models with synthetic chains-of-thought-and-action. *arXiv preprint arXiv:2412.05479*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. 2023. Gaia: a benchmark for general ai assistants. *arXiv preprint arXiv:2311.12983*.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2021. Metaicl: Learning to learn in context. *arXiv preprint arXiv:2110.15943*.

Xuanfan Ni, Hengyi Cai, Xiaochi Wei, Shuaiqiang Wang, Dawei Yin, and Piji Li. 2024. XL^2 bench: A benchmark for extremely long context understanding with long-range dependencies. *arXiv preprint arXiv:2404.05446*.

OpenAI. 2023. Gpt-4 system card.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366, Miami, Florida, USA. Association for Computational Linguistics.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2024. Training software engineering agents and verifiers with swe-gym. *arXiv preprint arXiv:2412.21139*.

Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2022. Grips: Gradient-free, edit-based instruction search for prompting large language models. *arXiv preprint arXiv:2203.07281*.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968, Singapore. Association for Computational Linguistics.

Cheng Qian, Chi Han, Yi R Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. 2023. Creator: Tool creation for disentangling abstract and concrete reasoning of large language models. *arXiv preprint arXiv:2305.14318*.

Mathieu Ravaut, Aixin Sun, Nancy Chen, and Shafiq Joty. 2024. On context utilization in summarization with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2764–2781, Bangkok, Thailand. Association for Computational Linguistics.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Linxin Song, Jiale Liu, Jieyu Zhang, Shaokun Zhang, Ao Luo, Shijian Wang, Qingyun Wu, and Chi Wang. 2024. Adaptive in-conversation team building for language model agents. *arXiv preprint arXiv:2405.19425*.

Hongjin Su, Jungo Kasai, Chen Henry Wu, Weijia Shi, Tianlu Wang, Jiayi Xin, Rui Zhang, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, et al. 2022. Selective annotation makes language models better few-shot learners. *arXiv preprint arXiv:2209.01975*.

Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. 2025. Mobile-agent-e: Self-evolving mobile assistant for complex tasks. *arXiv preprint arXiv:2501.11733*.

Zhiruo Wang, Daniel Fried, and Graham Neubig. 2024a. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks. *arXiv preprint arXiv:2401.12869*.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. 2024b. Agent workflow memory. *arXiv preprint arXiv:2409.07429*.

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

Yuxin Wen, Neel Jain, John Kirchenbauer, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2024. Hard prompts made easy: Gradient-based discrete optimization for prompt tuning and discovery. *Advances in Neural Information Processing Systems*, 36.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. 2024a. Longmemeval: Benchmarking chat assistants on long-term interactive memory. *arXiv preprint arXiv:2410.10813*.

Shirley Wu, Shiyu Zhao, Qian Huang, Kexin Huang, Michihiro Yasunaga, Kaidi Cao, Vassilis N Ioannidis, Karthik Subbian, Jure Leskovec, and James Zou. 2024b. Avatar: Optimizing llm agents for tool-assisted knowledge retrieval. *arXiv preprint arXiv:2406.11200*.

Zhiyong Wu, Yaoxiang Wang, Jiacheng Ye, and Lingpeng Kong. 2022. Self-adaptive in-context learning: An information compression perspective for in-context example selection and ordering. *arXiv preprint arXiv:2212.10375*.

Xiaobo Xia and Run Luo. 2025. Gui-r1: A generalist r1-style vision-language action model for gui agents. *arXiv preprint arXiv:2504.10458*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Jing Hua Toh, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. 2025. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Xunjian Yin, Xinyi Wang, Liangming Pan, Xiaojun Wan, and William Yang Wang. 2024. G\" odel agent: A self-referential agent framework for recursive self-improvement. *arXiv preprint arXiv:2410.04444*.

Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi R Fung, Hao Peng, and Heng Ji. 2023. Craft: Customizing llms by creating and retrieving from specialized toolsets. *arXiv preprint arXiv:2309.17428*.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. 2023. Self-taught optimizer (stop): Recursively self-improving code generation. *arXiv preprint arXiv:2310.02304*.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2024. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077, Bangkok, Thailand. Association for Computational Linguistics.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, et al. 2024a. Aflow: Automating agentic workflow generation. *arXiv preprint arXiv:2410.10762*.

Shaokun Zhang, Xiaobo Xia, Zhaoqing Wang, Ling-Hao Chen, Jiale Liu, Qingyun Wu, and Tongliang Liu. 2023. Ideal: Influence-driven selective annotations empower in-context learners in large language models. *arXiv preprint arXiv:2310.10873*.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. 2025. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. In *Forty-second International Conference on Machine Learning*.

Shaokun Zhang, Jieyu Zhang, Dujian Ding, Mirian Hipolito Garcia, Ankur Mallick, Daniel Madrigal, Menglin Xia, Victor Rühle, Qingyun Wu, and Chi Wang. 2024b. Ecoact: Economic agent determines when to register what action. *arXiv preprint arXiv:2411.01643*.

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

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2024d. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Zihao Zhao, Eric Wallace, Shi Feng, Dan Klein, and Sameer Singh. 2021. Calibrate before use: Improving few-shot performance of language models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12697–12706. PMLR.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.

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

A More Experiments and Analysis

How efficient is FGO in terms of optimizing time consumption? In this section, we present the time to optimize using each method. The experimental results demonstrate that FGO achieves superior optimization efficiency. As shown in Figure 7, FGO consistently delivers the highest accuracy while requiring significantly less training time compared to alternatives. Notably, on ALFWorld, FGO achieves 85% accuracy in ~2000 seconds, substantially outperforming All-at-once optimization by 4 times. The key to FGO's efficiency lies in its divide-and-conquer approach, which enables parallel optimization of dataset subsets rather than sequential processing. By distributing the computational load and then adaptively merging results, FGO effectively bypasses the scaling limitations of other optimization methods.

How does FGO perform with advanced partitioning strategy? In this section, we investigate the effectiveness of a more sophisticated data partitioning strategy on FGO's performance. We develop a strategy that partitions tasks based on inherent task difficulty. To achieve this, we performed an initial agent rollout on ALFWorld training tasks and used the resulting trajectory length as a proxy for difficulty. Tasks that required longer trajectories or reached the maximum step limit were considered more difficult. Subsequently, all tasks were sorted by this metric and grouped into partitions of comparable difficulty. As shown in Table 5, difficulty-based partitioning reaches a competitive result with our original category-based partitioning. This result is significant because it confirms that FGO is a flexible framework that can benefit from various partitioning strategies. It also reinforces the conclusion that the progressive merging algorithm is the primary driver of performance, as it successfully synthesizes optimal modules even when the partitioning logic changes.

Scenario	ALFWorld	
	Avg of 3	Best of 3
Category	83.6	**89.6**
Difficulty	**84.0**	**89.6**

Table 5: Results comparison between difficulty-aware partitioning and category-based partitioning.

B Experiment Details

B.1 Baselines

We provide a detailed description of the baselines compared in the main experiment here.

B.1.1 Prompt Optimization Benchmarks

Reflexion is a self-refinement-based iterative method that uses LLM to provide verbal feedback after each task solving attempt. In this setting, an LLM agent repeatedly attempts a task. If the attempt fails, an external LLM will be prompted to generate reflections on reasons why the task fails, and the reflection will append to the context for prompting the LLM agent in the next attempt. Reflexion operates on task-basis, where a reflection is learned for every task.

ProTeGi is an automatic prompt optimization technique inspired by gradient descent, designed to iteratively refine prompts through discrete textual edits. It operates by generating natural language "gradients," or feedback on current prompt errors, which guide prompt improvements in the semantic opposite direction. These prompt candidates are systematically explored using beam search combined with efficient bandit-based selection.

MIPROv2 is a meta-optimization-based optimizer tailored for multi-stage Language Model programs. MIPROv2 factorizes prompt optimization into crafting effective instructions and demonstrations for each program module. It addresses two central challenges: prompt proposal, by using bootstrapped demonstrations and contextually informed instructions, and credit assignment, through Bayesian optimization to jointly optimize multiple latent prompt parameters efficiently. The optimization process is conducted iteratively, where candidate instructions and demonstrations are proposed, evaluated using stochastic mini-batch validation, and refined through meta-optimization. MIPROv2 learns generalizable instructions applicable across multiple tasks, rather than task-specific prompts, making it a versatile online optimization method for complex LM programs.

B.1.2 Tool Optimization Benchmarks

CRAFT is a method that enhances LLM performance by creating specialized toolsets for specific tasks. It generates diverse code snippets by prompting advanced LLMs to solve sampled problems, abstracts these snippets for broader usability, and applies validation and deduplication to ensure correct-

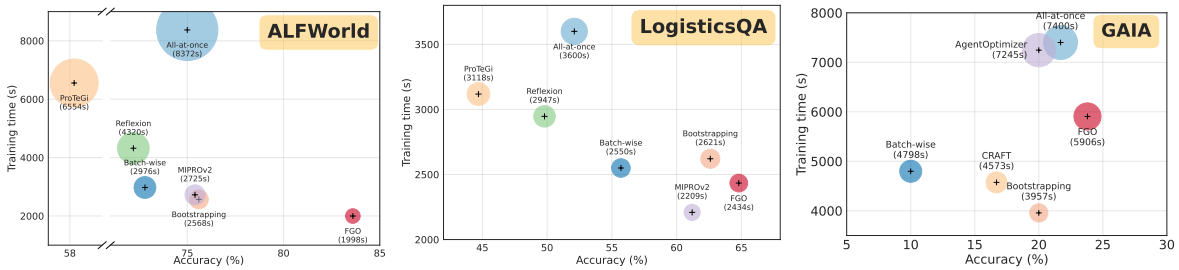

Figure 7: Comparison of time to optimize using different optimization methods on ALFWorld, LogisticsQA, and GAIA. Each panel plots the trained agent's performance against the time to optimize the agentic system. Circle diameters are proportional to the optimization time consumption, with crosses (+) indicating circle centers.

ness and reduce redundancy. At inference, CRAFT employs a multi-view matching strategy, considering problem context, API names, and docstrings, to dynamically retrieve relevant tools from the customized toolsets. CRAFT constructs the toolsets offline but retrieves tools online during inference.

AgentOptimizer is an offline agent training method that treats tools used by LLM agents as learnable parameters. Instead of tuning the LLM itself, AgentOptimizer optimizes the agent's tool set via a training loop that evaluates execution histories and iteratively updates functions using LLM-based suggestions. At each step, it proposes edits—adding, revising, or removing functions—based on task performance. This process learns a general set of functions applicable across tasks, rather than per-task customizations.

B.1.3 Proposed Agent Optimization Baselines

All-at-Once strictly implements the algorithm 1. In one epoch, it prompts the agent with the whole training dataset, then collect the trajectory and feedback, perform LLM-based optimization to obtain the updated module. Note that this can encounter context window overflow error when the trajectory is long or training dataset size is large.

Batch-wise is a workaround to the context window overflow error. As detailed in Algorithm 3, this approach divides the full dataset $\mathcal{D}$ into smaller mini-batches at the beginning of each epoch. For every mini-batch, the agent generates trajectories for each task within the batch, collects the corresponding feedback, and then the module is updated using only the history from that specific mini-batch. This iterative process of processing and updating per mini-batch can significantly reduce the amount of information that needs to be handled at any single optimization step, thereby mitigating the risk of exceeding the LLM's context window limit.

Algorithm 3 Batch-wise Module Optimization

Input: Task set $\mathcal{D}$, num epochs E, batch size B
Output: Optimized module θ
$\theta \leftarrow \phi$ ▷ Init θ
for $e \leftarrow 1$ to E **do**
  Batches $\leftarrow$ split($\mathcal{D}$) ▷ Split $\mathcal{D}$ into batches
  for $\mathcal{D}_{\text{batch}} \in$ Batches **do** ▷ Perform optimization on each batch
    $\mathcal{H}_{\text{batch}} \leftarrow \{\}$
    for $(q, y) \in \mathcal{D}_{\text{batch}}$ **do**
      $\tau \leftarrow \mathcal{A}(q; \theta)$ ▷ Eq. 1
      $r \leftarrow \text{Evaluate}(\tau, y)$
      $\mathcal{H}_{\text{batch}}.\text{append}((\tau, r))$
    end for
    $\theta \leftarrow \text{LLM}_{\textbf{optim}}(\mathcal{H}_{\text{batch}}, \theta)$
  end for
end for
return θ

Bootstrapping is an alternative approach to mini-batch processing that aims to better approximate the overall task distribution throughout training. As shown in Algorithm 4, instead of partitioning the training dataset into fixed mini-batches at the start of an epoch, this method samples mini-batches from the entire dataset $\mathcal{D}$ **with replacement** for a predefined number of iterations. In each iteration, a new mini-batch $\mathcal{D}_{\text{sample}}$ is drawn. The agent then updates module based on execution results on the mini-batch. The key advantage of this approach is that by continuously sampling with replacement, each mini-batch has the potential to draw from the full diversity of the dataset in every iteration, potentially leading to a more robust and generalized module update. This can be particularly beneficial when the dataset exhibits a complex or uneven distribution of task types.

Algorithm 4 Bootstrapping Module Optimization

Input: Task set $\mathcal{D}$, num iterations N, batch size B
Output: Optimized module θ
$\theta \leftarrow \phi$ ▷ Init θ
for $i \leftarrow 1$ to N **do**
  $\mathcal{D}_{\text{sample}} \leftarrow \text{Sample}(\mathcal{D}, B)$ ▷ Sample a batch with replacement
  $\mathcal{H}_{\text{sample}} \leftarrow \{\}$
  for $(q, y) \in \mathcal{D}_{\text{sample}}$ **do**
    $\tau \leftarrow \mathcal{A}(q; \theta)$ ▷ Eq. 1
    $r \leftarrow \text{Evaluate}(\tau, y)$
    $\mathcal{H}_{\text{sample}}.\text{append}((\tau, r))$
  end for
  $\theta \leftarrow \text{LLM}_{\textbf{optim}}(\mathcal{H}_{\text{sample}}, \theta)$
end for
return θ

B.2 Implementation Details

We present the hyperparameters used in the main experiments in Table 6. We fix the cluster size threshold t to 3. We use 'text-embedding-3-large' model from OpenAI to calculate the embeddings of the modules.

For ALFWorld, the training tasks are randomly drawn from the 'train' split, and the evaluation tasks are from the 'valid_unseen' split. The tasks belongs to one of the six different categories, i.e. 'Pick', 'Clean', 'Heat', 'Cool', 'Look', 'Pick 2'. We divide tasks based on their category in the task-division stage.

For GAIA, since their 'test' split does not disclose ground truth and therefore we cannot evaluate on it, we select both the training and evaluation tasks from the 'validation' split randomly. We divide tasks randomly in the task-division stage for GAIA and LogisticsQA.

To control the generation of imbalanced partitioning as mentioned in the ablation study, we use a hyperparameter β. In the context of ALFWorld, the number of samples in each subset n_i is determined by the following equation:

$$n_i = \frac{60\beta^i}{\sum_{j=1}^{6} \beta^j}$$

The closer β is to 1, the more balanced the partitioning. We explored two β values: 0.6 and 0.8, where the numbers of samples per subset are [25, 15, 9, 6, 3, 2] and [16, 13, 11, 8, 7, 5] respectively. When β is 0.6, the partitioning is highly imbalanced, many subsets lack comprehensive coverage of all task subtypes, making optimization on such subsets more challenging.

C Case Studies

C.1 Failure Analysis

In this section, we provide a case study on the failure analysis of FGO on ALFWorld. We analyze a case where imbalanced partitioning leads to suboptimal performance.

When $\beta = 0.6$, the number of samples in each subset are [25, 15, 9, 6, 3, 2]. This results in subsets lacking comprehensive coverage of all task subtypes. Consequently, the prompts optimized on these subsets suffer from incomplete knowledge of the entire task landscape, leading to lower-quality and overly specialized prompts. In the subset with 3 samples, the learned workflow in the prompt is way too specific. Content as follows:

```
# Workflow for 'clean' type task
Step 1: Locate kettle
- THOUGHT: Identify probable placements
- ACTION: take kettle from [loc]
Step 2: Use sinkbasin to clean kettle
- ACTION: use sinkbasin
Step 3: Open cabinet
- ACTION: open cabinet
Step 4: Put kettle in cabinet
- ACTION: put kettle in cabinet

# Hints
You usually skip the cleaning stage.
Always clean the item with:
"use [sinkbasin]"
```

This instruction overfits to a specific training task, summarizing the task rather than generalizing across tasks of the same subtype. Although the progressive merging stage partly rectifies this by merging instructions from other subsets, the final synthesized instruction still misses critical generalized elements: going to the sinkbasin first before cleaning any item. This missing step is one of the core reasons why the agent fails. We hypothesize that this oversight occurs because the highly uneven data distribution complicates the identification of subtle yet significant task-solving patterns.

C.2 Comparison with Baseline

To demonstrate the practical advantages of FGO, we present a case study focusing on the web-related tools it optimized on GAIA.

Table 6: Hyperparameter of Different Datasets

Dataset	# Train	# Test	# Splits	Epoch	Agent Backbone	Optimizer Backbone
ALFWorld	60	134	6	4	gpt-4o-mini	gpt-4o
LogisticsQA	48	219	8	5	gpt-4o	gpt-4o
GAIA	36	60	4	4	gpt-4o	gpt-4o

FGO-Optimized Tool. Our method generates a robust and comprehensive toolset. The search_bing function includes error handling and a count parameter for flexible results. It is complemented by a fetch_web_page_content tool that intelligently extracts specific, relevant information from the retrieved URLs.

```
import os
import requests
from bs4 import BeautifulSoup

def search_bing(query, count=10):
    search_api_key = os.getenv('
    BING_SEARCH_V7_SUBSCRIPTION_KEY')
    search_endpoint = os.getenv('
    BING_SEARCH_V7_ENDPOINT').rstrip('/') + "/v7
    .0/search"

    headers = {
        'Ocp-Apim-Subscription-Key':
    search_api_key,
        'Content-Type': 'application/json'
    }
    params = {
        'q': query,
        'textDecorations': True,
        'textFormat': 'HTML',
        'count': count
    }

    try:
        response = requests.get(search_endpoint,
     headers=headers, params=params)
        response.raise_for_status()
        return response.json()
    except requests.exceptions.HTTPError as
    http_err:
        return {'error': f'HTTP error occurred:
    {http_err}'}
    except Exception as err:
        return {'error': str(err)}

def fetch_web_page_content(url):
    try:
        response = requests.get(url, verify=
    False)  # Disable SSL verification for
    inaccessible certs

        if response.status_code == 200:
            soup = BeautifulSoup(response.
    content, 'html.parser')
            if soup.script:
                [s.extract() for s in soup('
    script')]
            if soup.style:
                [s.extract() for s in soup('
    style')]
            tables = soup.find_all('table')
            for table in tables:
                if table.find('th') and "Number"
     in table.find('th').text:
                    return str(table)
            return str(soup)
        else:
            return "Failed to retrieve content"

    except requests.exceptions.RequestException
    as e:
        return f"An error occurred: {e}"
```

Baseline Tool We select a representative example from baselines for comparison. In contrast, Batch-wise optimization produces a functionally limited tool. It lacks error handling and offers no flexibility in the number of search results returned.

```
import os
import requests

def search_bing(query):
    """
    Executes a basic Bing search with minimal
    configuration.
    """
    search_api_key = os.getenv('
    BING_SEARCH_V7_SUBSCRIPTION_KEY')
    endpoint = os.getenv('
    BING_SEARCH_V7_ENDPOINT').rstrip('/') + "/v7
    .0/search"

    headers = {'Ocp-Apim-Subscription-Key':
    search_api_key}
    params = {'q': query}

    response = requests.get(endpoint, headers=
    headers, params=params)
    response.raise_for_status()
    return response.json()
```

In comparison, FGO-generated tools are demonstrably superior for real-world use:

- Robustness: Our tool handles common network or API errors gracefully, whereas Batch-wise optimized tool would crash.
- Completeness: FGO produces an end-to-end solution that not only searches but also intelligently extracts information, a critical step that the baseline method neglects entirely.

This example demonstrates how FGO produces more reliable and functionally complete tools for

agentic systems.

D Visualization of the Merging Process

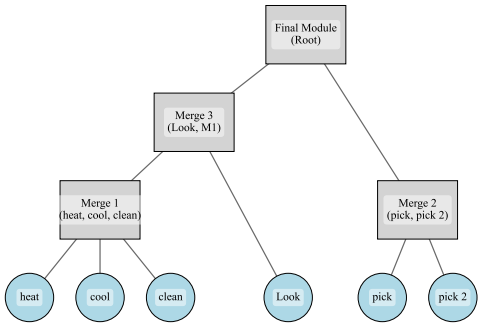

Figure 8: The visualization of the merging tree.

We visualize one of the merging trees in Figure 8. As shown in the figure, the dynamic merging process can identify similar modules from different partitions and merge accordingly. The prompt trained on 'heat', 'cool' and 'clean' are identified as similar, and merged into a single prompt. This is coherent with the task nature, as these tasks' workflow all involve identifing the object location, perform some action on it, and then put it to a target location. 'pick' and 'pick 2' are also merged at the same time, as the two type of tasks are inherently similar. Overall, the merging tree shows that FGO can effectively identify similar modules and merge them into a single one.

E LogisticQA Dataset

E.1 Background

We evaluate our system on a collection of real-world Universal Business Language invoice documents, developed in cooperation with one of the world's largest logistics companies. The primary task is to extract transport reference numbers from these documents. The reference numbers exist in these invoice documents in a non-fixed pattern. It typically requires human effort to extract it manually during real-world business operations. AI agents that can effectively understand the context and extract reference numbers can make the business workflow more efficient. The LogisticQA dataset shows LLMs' ability to achieve such a goal. It contains 267 valid invoice documents and transport reference pairs. It can also reflect LLM's instruction-learning capability in real-world document understanding tasks.

The dataset presents several challenging characteristics that make it an ideal testbed for evaluating the instruction learning capabilities. First, it requires specialized domain knowledge of business documents and terminology not commonly found in general language model training. Second, the hierarchical structure of UBL documents and the significant variability in format and identification patterns pose substantial extraction challenges. Additionally, as a novel benchmark without prior literature coverage, this dataset offers unique opportunities to assess agents' adaptive learning abilities in a practical, high-stakes business context.

E.2 Dataset Statistics

The analysis of our XML business document dataset demonstrates strong alignment with real-world business documentation patterns, as shown in Figure 9. The document length distribution peaks between 200-500 lines, while the XML structure complexity with most documents containing 100-400 tags. The token distribution centered around 2,000-4,000 tokens indicates a long-context understanding challenge for LLMs. Notably, the language distribution across documents (Turkish: 39.5%, English: 29.6%, Spanish: 22.0%, Italian: 8.9%) reflects a realistic multinational business environment, particularly common in European and Mediterranean operations where English serves as a lingua franca alongside regional languages.

E.3 Dataset Example

Here is an example XML business document in the dataset. The ground truth extraction is `847 5321 9084`. The named and loations in the dataset are all anonymized.

```
<?xml version="1.0" encoding="UTF-8"?>
<Invoice xmlns="urn:oasis:names:specification:ubl:schema:xsd:
      Invoice-2"
        xmlns:cac="urn:oasis:names:specification:ubl:schema:
             xsd:CommonAggregateComponents-2"
        xmlns:cbc="urn:oasis:names:specification:ubl:schema:
             xsd:CommonBasicComponents-2">
    <cbc:UBLVersionID>2.1</cbc:UBLVersionID>
    <cbc:CustomizationID>urn:cen.eu:en16931:2017#compliant#urn:
         fdc:peppol.eu:2017:poacc:billing:3.0</cbc:
         CustomizationID>
    <cbc:ID>rmCMsB6Km6J4Qp2a</cbc:ID>
    <cbc:IssueDate>2023-10-11</cbc:IssueDate>
    <cbc:InvoiceTypeCode>Invoice</cbc:InvoiceTypeCode>
    <cbc:DocumentCurrencyCode>TRY</cbc:DocumentCurrencyCode>

    <cbc:Note>SALE
HADIMKOY BRANCH 847 5321 9084
No withholding tax applies when not self-owned according to
      law
This invoice must be paid by: 01/08/24
PLEASE INDICATE THE VEHICLE PLATE NUMBER AND INVOICE NUMBER IN
       THE DESCRIPTION OF YOUR BANK TRANSFER RECEIPT
For invoices not paid by due date, late payment interest will
      be charged according to the Law on Collection Procedure
      of Public Receivables (AATUHK).
```

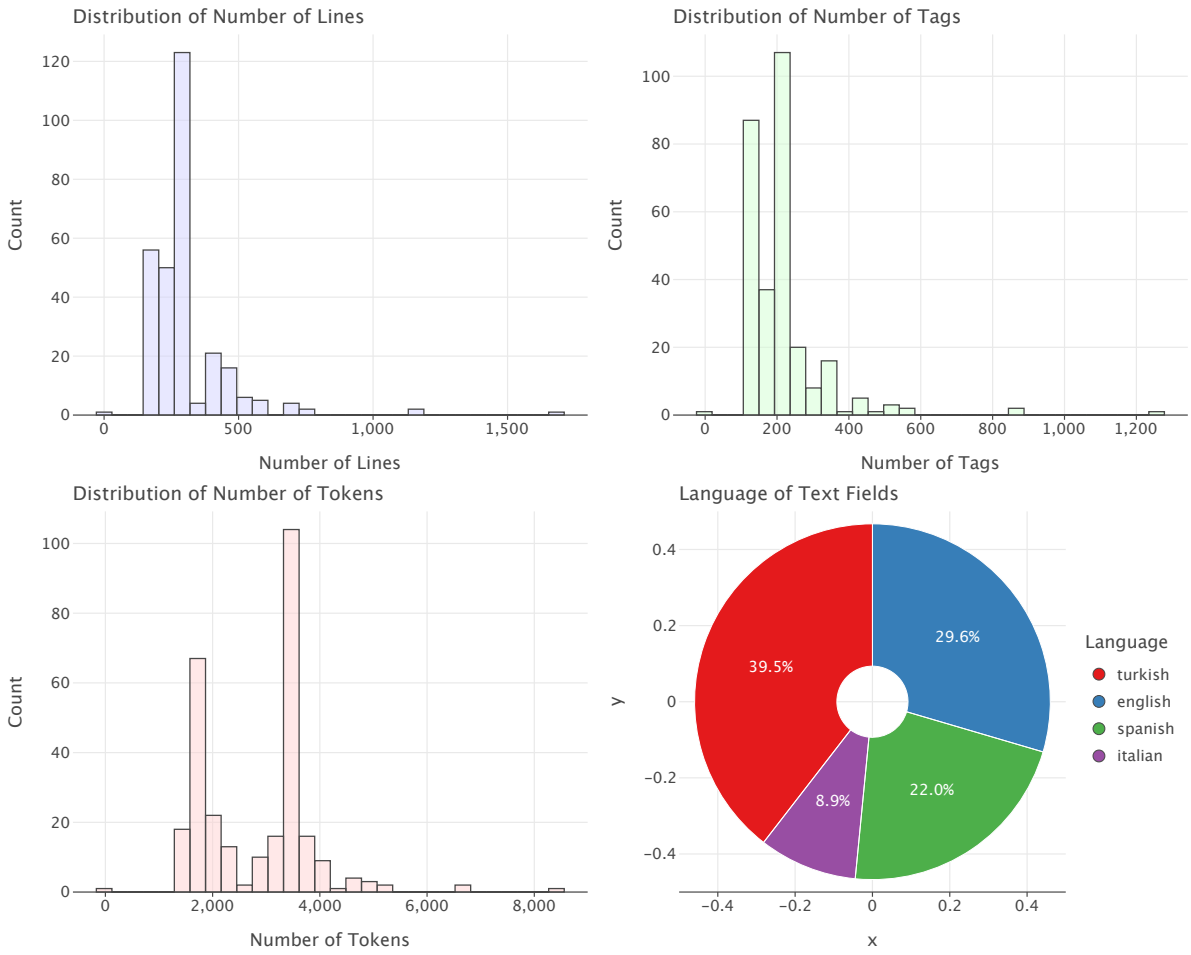

Figure 9: Statistical analysis of XML business documents. Top left: Distribution of document lengths showing typical business document sizes. Top right: Distribution of XML tags indicating document structure complexity. Bottom left: Token distribution demonstrating the long context challenge for LLM. Bottom right: Language distribution across documents reflects business documents' multinational nature.

```
Only FourThousandThirtyTwoTL</cbc:Note>


  <cac:AccountingSupplierParty>
    <cac:Party>
      <cac:PartyName>
        <cbc:Name>S.S 350 COOPERATIVE AIRPORT CARGO
            TERMINAL LOGISTICS SERVICES MOTOR CARRIERS
            </cbc:Name>
      </cac:PartyName>
      <cac:PostalAddress>
        <cbc:StreetName>Cargo Terminal Cooperative
            Service</cbc:StreetName>
        <cbc:CityName>Springfield</cbc:CityName>
        <cbc:PostalZone>None</cbc:PostalZone>
        <cac:Country>
          <cbc:IdentificationCode>TR</cbc:
              IdentificationCode>
        </cac:Country>
      </cac:PostalAddress>
    </cac:Party>
  </cac:AccountingSupplierParty>

  <cac:AccountingCustomerParty>
    <cac:Party>
      <cac:PartyName>
        <cbc:Name>GLOBAL LOGISTICS SOLUTIONS LTD.</cbc:
            Name>
      </cac:PartyName>
      <cac:PostalAddress>
        <cbc:StreetName>INDUSTRIAL DISTRICT SPRINGFIELD<
            /cbc:StreetName>
        <cbc:CityName>None</cbc:CityName>
        <cbc:PostalZone>None</cbc:PostalZone>
        <cac:Country>
          <cbc:IdentificationCode>TR</cbc:
              IdentificationCode>
        </cac:Country>
      </cac:PostalAddress>
    </cac:Party>
  </cac:AccountingCustomerParty>


    <cac:PaymentTerms>
      <cbc:Note>SALE
HADIMKOY BRANCH 847 5321 9084
No withholding tax applies when not self-owned according to
    law
This invoice must be paid by: 01/08/24
PLEASE INDICATE THE VEHICLE PLATE NUMBER AND INVOICE NUMBER IN
     THE DESCRIPTION OF YOUR BANK TRANSFER RECEIPT
For invoices not paid by due date, late payment interest will
    be charged according to the Law on Collection Procedure
    of Public Receivables (AATUHK).
Only FourThousandThirtyTwoTL</cbc:Note>
    </cac:PaymentTerms>


    <cac:LegalMonetaryTotal>
      <cbc:LineExtensionAmount currencyID="TRY">
        2243.26
      </cbc:LineExtensionAmount>
      <cbc:TaxExclusiveAmount currencyID="TRY">
        448.65
      </cbc:TaxExclusiveAmount>
      <cbc:TaxInclusiveAmount currencyID="TRY">
```

```
            2691.91
        </cbc:TaxInclusiveAmount>
        <cbc:PayableAmount currencyID="TRY">
            2691.91
        </cbc:PayableAmount>
    </cac:LegalMonetaryTotal>


    <cac:InvoiceLine>
        <cbc:ID>1</cbc:ID>
        <cbc:InvoicedQuantity unitCode="EA">1.0</cbc:
            InvoicedQuantity>
        <cbc:LineExtensionAmount currencyID="TRY">
            2243.26
        </cbc:LineExtensionAmount>
        <cac:Item>
            <cbc:Description>THY-NEWTOWN transportation fee-78
                XYZ432</cbc:Description>
            <cbc:Name>THY-NEWTOWN transportation fee-78XYZ432</
                cbc:Name>
        </cac:Item>
        <cac:Price>
            <cbc:PriceAmount currencyID="TRY">2243.26</cbc:
                PriceAmount>
        </cac:Price>
    </cac:InvoiceLine>

</Invoice>
```


F Complexity Analysis

In this section, we analyze the computational complexity of the recursive clustering in the progressive merging process.

F.1 Clustering Tree Depth

At each recursive step, the number of module is reduced by taking the square root:

$$n_{i+1} = \sqrt{n_i}, \quad \text{with } n_0 = N. \tag{3}$$

The recursion stops when the number of items satisfies:

$$n_D = N^{(1/2)^D} \leq t. \tag{4}$$

Taking logarithms on both sides gives:

$$(1/2)^D \cdot \log N \leq \log t. \tag{5}$$

Solving for D yields:

$$D = O\left(\log \log N\right). \tag{6}$$

F.2 Backtesting Complexity

Each merge operation performs a backward testing over all tasks contributing to the merged module. Since tasks are merged without duplication, the total number of unique tasks remains T throughout the process. As every level of the clustering tree processes T tasks and the depth of the tree is $D = O(\log \log N)$, the overall complexity of testing is:

$$O\left(T \cdot \log \log N\right). \tag{7}$$

This demonstrates that the overhead introduced by backward testing is modest as N scales.

G Prompt

G.1 ALFWorld

> Perform actions and interact with a household to solve a task. At the beginning of your interactions, you will be given the detailed description of the current environment and your goal to accomplish. The environment only accept certain format of actions. Here are two examples, learn the pattern carefully.
> {example}

Figure 10: Prompt for agent solving ALFWorld.

G.2 LogisticsQA

> # Task background
> Read the content of a xml file which contains a shipment invoice document in UBL format. You are tasked to understand the content and extract the transport reference number from it.
> When you reach a conclusion, format your answer as "final answer: [extracted reference number]"

Figure 11: Prompt for agent solving LogisticsQA.

G.3 GAIA

> You need to solve the question below given by a user. When you are solving tasks, explicitly consider whether the task can benefit from web navigation capability.
> # Task
> {task}

Figure 12: Prompt for agent solving GAIA.

H Potential Risks

We use close source API in this research, which can cause trouble on private dataset not intentioned for release, leading to potential privacy leakage if there is misconduct in API providers.

You are an AI assistant specialized in optimizing guidelines. Your goal is to analyze various inputs and improve the current guideline. You will be presented with the logs of solving similar tasks, the current guideline, the performance of the current guideline in term of success rate, and the historic guidelines and respective performance.
Background:
TASK_DESCRIPTION

Please review the following information:
1. Expert Demonstration:
Below is expert solution that demonstrates the correct approach to similar task.
EXAMPLE_TRAJECTORY
2. Agent Logs
Here are the logs of how the agent apply the current guideline to solve the task.
LOGS
3. Past Guidelines
Here are the previously proposed guidelines and their respective performance.
PAST_STATISTICS
4. Current Guideline
Below is the current guideline and its success rate:
GUIDELINE

Your task is to analyze the provided information and create improved guidelines. Before formulating your final guidelines, provide your analysis of the current task. In this process:
a. Analyze the action sequence from the trajectories
- For a successful attempt, extract the general workflow of how it is done.
- For a failed attempt, analyze how it deviates from the successful examples and how it can be summarized into a rule to avoid similar failures.
b. Identify patterns and failure modes
- Go through the whole set of logs, list recurring successful strategies, with specific examples.
- Enumerate common mistakes made by the agents.
c. Evaluate current guidelines
- Assess the effective parts of the current guideline, how it contributes to agents correctly solving the task, with supporting examples
- Identify areas for improvement, which rule is not covered by the current guideline, with specific examples.
d. Refine valid action space
- List all valid actions found in input data and generalize actions into categories

Be thorough and detailed in your analysis, using specific examples from the input data to support each point. It's OK for this section to be quite long.

Your output should be a JSON object. Below is an example.
SAMPLE_OUTPUT

Figure 13: Prompt for LLM optimizer to optimize instructions.

Background
You are a guideline synthesizer tasked with merging multiple expert-generated guidelines for solving tasks from the same distribution. Your role is to identify common patterns across different expert perspectives while preserving unique valuable insights. You will analyze multiple guidelines along with their performance statistics to create a comprehensive unified guideline that maintains high generalizability.
TASK_DESCRIPTION

Below are some principles to follow when merging the guidelines:
1. The guidelines all follow structured format. Keep the structure as it as and merge the field of each guideline correspondingly.
2. The guidelines come with their performance tested on different tasks. Weight patterns from guidelines with higher total attempts more heavily, particularly when success rates are comparable.

Expert Guidelines and Statistics
STATISTICS

Output Format
Your response should be in markdown with two sections:

> # Analysis
> Use this section to document your reasoning in synthesis:
> - Analyze the common and difference between different tasks
> - Common patterns identified across expert guidelines
> - Unique valuable insights from individual experts
> - Statistical analysis of pattern effectiveness
> - Reasoning for inclusion/generalization decisions
>
> # Guidelines
> Put your synthesized guidelines here, the structure of your guideline should follow the originals.

Figure 14: Prompt for merging the instructions.

You are a tool synthesizer tasked with merging and improving multiple expert-generated toolboxes for answering general questions from users. Your goal is to create a comprehensive unified toolbox that maintains high generalizability while being most useful.
Context:
TASK_DESCRIPTION
To provide you with background on what the task solution looks like, here is an example trajectory:
EXAMPLE_TRAJECTORY
Here are the expert proposed guidelines and their performance statistics:
STATISTICS
Your task is to analyze these toolboxes, statistics, and the example trajectories to create a unified set of toolboxes. A toolbox contains a list of functions. One function signature includes the following five elements:
1. Function name
2. Function description
3. JSON schema of arguments encoded as a string
4. A list of package names imported by the function
5. The code implementation

This is an example of a function that might worth add:
EXAMPLE_FUNCTION
You should comprehensively consider all the toolboxes proposed by experts. If functions have similar functionality, write a function that generalizes most.
Before providing your final output, show your thought process and reasoning. This should include:
a. Evaluate the similar functions.
b. Evaluate the functions proposed by different experts based on the performance, accountability and corner case.
c. An outline of the decision-making process for arriving at the final guides.
d. Identify whether the functions can be improved, like adding more detailed if-else, can be generalized to more cases, etc. Return the improved function.

It's OK for this section to be quite long.

After your analysis, you MUST respond with JSON object in the following format. The 'response' field will contain the list of all the merged and improved toolbox.

```
{
    "analysis": "Your detailed analysis goes here.",
    "response": [
        {
            "name": "your proposed function1 name",
            "description": "",
            "arguments": {
            },
            "packages": "",
            "code": ""
        },
    ]
}
```

Figure 15: Prompt for merging the tools.

You are an AI tasked with analyzing and reflecting on trajectories. Your goal is to evaluate an agent's performance, compare it to successful demonstrations, and provide insights for improvement.

First, review these two successful demonstrations of completing the task: DEMO Now, examine the following trajectory attempted by an agent: TRAJECTORY Please reflect on the agent's trajectory and determine if it is correct.
If the agent's trajectory is correct:
- Analyze the workflow of successfully solving the task.
If the agent's trajectory is incorrect:
- Identify the point of deviation from the successful demonstrations
- Explain the key error(s) made by the agent
- Describe how this error can be avoided

Based on your analysis, propose a suggestion to help the agent avoid making the same mistake in the next trial or to further improve their performance.

Below is an example of the trajectory and the reflection process.
REFLECTION_EXAMPLE

Please structure your response with two tags as follows, the part in `<reflection>` tag will be used to guide the next trial:
`<analysis>` your analysis of the trajectory here`</analysis>`
`<reflection>` your formal reflection here`</reflection>`

Figure 16: Prompt for evaluating the trajectory.