Divide, Optimize, Merge: Fine-Grained LLM Agent Optimization at Scale

Anonymous ACL submission

Abstract

LLM-based optimization has shown remarkable potential in enhancing agentic systems. However, the conventional approach of prompting LLM optimizer with the whole training trajectories on 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 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 demonstrate that FGO outperforms existing approaches 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 optimization of increasingly sophisticated agent systems. Further analysis demonstrates that FGO achieves the most consistent performance gain in all training dataset sizes, showcasing its scalability and efficiency.

1 Introduction

Large Language Models (LLMs) have emerged as powerful optimizers for LLM 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), tools (Qian et al., 2023; Zhang et al., 2024c,b; Wang et al., 2024a). These agentic systems have shown promising results in enhancing agent performance across various domains, including reasoning (Cheng et al., 2024; Zelikman et al., 2023), software engineering (Jimenez et al., 2023; Pan et al., 2024), data analysis (Hu et al., 2024b; Jing et al., 2024), computer using (Wang et al., 2025; Xie et al., 2025; Abuelsaad et al., 2024).

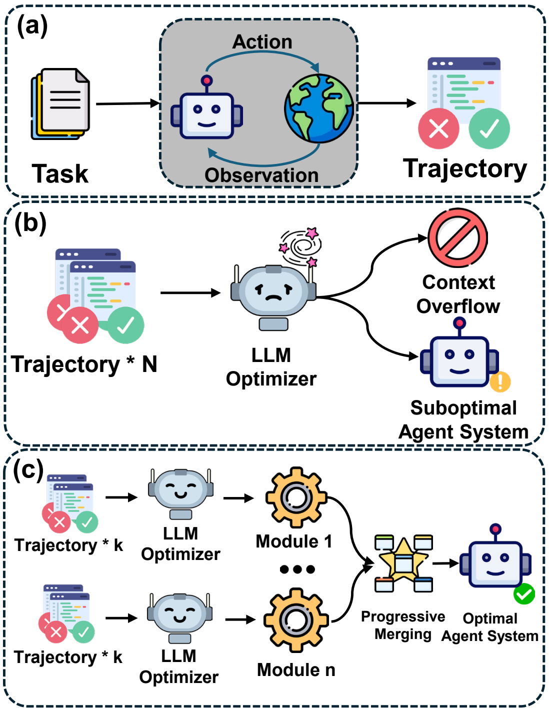

Figure 1: Agent optimization approaches. (a) Basic agent execution process. (b) Traditional all-at-once optimization faces context overflow and inferior performance with large trajectory data. (c) Our method: divide-and-conquer optimization with progressive merging enables scalable processing of large datasets.

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, it becomes problematic as the data grows. For instance, in the GAIA benchmark (Mialon et al., 2023), agents normally rely on external tools to collect real-world information and generate lengthy execution traces for subsequent optimization, which is filled with raw documents and complex intermediate reasoning steps, even challenge for human to parse. 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 extensivecorpus (Bai et al., 2024; Liu et al., 2024; Ni et al., 2024; Ravaut 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 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 optimization enabling efficient processing of each subset, and (3) Progressive module merging that adaptively combines 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 agent could access. Across diverse tasks including ALFWorld (Shridhar et al., 2020), LogisticsQA, and GAIA (Mialon et al., 2023). Agent trained with FGO produces 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 demonstrate FGO's effectiveness across diverse tasks 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 meaningfully influence the agent's policy gradients when exposed to specific queries. This ensures the module is sufficiently responsive to reward signals during optimization. Second, the *orthogonality* property mandates 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. Given parameters θ, the agent's policy determines actions based on the current state of interaction and observation. These actions along with the observations form a trajectory τ that represents the agent's solution attempt for the given query.

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

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 prompt for task handling (Wen et al., 2024; Wu et al., 2024b), long term memory (Zhang et al., 2024e), the available toolbox (Zhang et al., 2024c), the weights of backbone LLM (Zeng et al., 2024; Ma et al., 2024).

2.2 Motivation

In LLM-as-optimizer setting, we assume the numeric value of the policy gradient is not accessiblein Eq. 2. This constraint emerges from a practical reality in modern LLM agent systems - the increasing reliance on proprietary Large Language Models 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 LLM context windows is a critical bottleneck in optimization. As training samples grow, the concatenated trajectories and module-loss pairs can exceed the context capacity of even the most capable LLMs. This limitation becomes particularly acute in complex tasks where individual trajectories contain extensive reasoning steps or multi-turn interactions. In such scenarios, even a modest number of samples can overwhelm the context window, severely limiting the LLM optimizer's ability to process comprehensive training data.

Insufficient context utilization. Even when the content fits within context limits, LLMs can face significant challenges in effectively processing and discovering patterns across extensive collections of trajectories (Ni et al., 2024). Recent benchmarks on long-form text comprehension and summarization tasks have consistently demonstrated that LLM's performance deteriorates significantly with increasing text length, particularly in processing complex dialogues and lengthy documents (Bai et al., 2024; Wu et al., 2024a; Ni et al., 2024; Song et al., 2024; Zhang et al., 2024d). In the context of LLM based optimizers, optimization requires grasping long-range dependencies and analyzing fine-grained details to capture subtle patterns across multiple lengthy samples. This inherent limitation of LLMs can lead to suboptimal module updates that fail to capture the full complexity of the optimization problem, especially in real-world applications where performance depends on understanding both broad patterns and fine-grained details across diverse samples.

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 begin with describing how we perform agent optimization. The pipeline is illustrated in Algorithm 1. In each epoch, the agent undergoes a three-phase cycle: exploration, evaluation, and optimization. During exploration, the agent interacts with the given task with the current module, generating the solution trajectories. The evaluation phase introduces a post-hoc LLM based evaluator that analyzes these trajectories to determine correctness, identify failures, patterns as well as potential areas for improvement based on the ground truth and trajectory. The evaluations serve as textual gradients to guide the direction for updating the instruction toward better performance. The optimization phase then leverages these insights by feeding the trajectories, textual gradients into an LLM based optimizer, which synthesizes this information to generate an updated module.

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

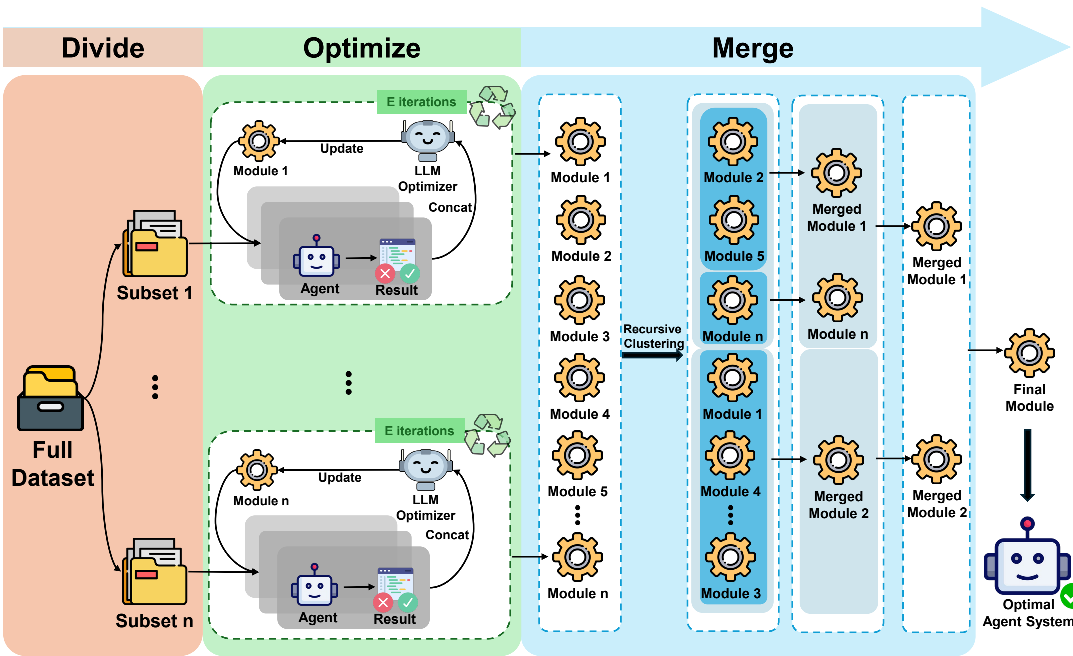

Figure 2: Illustration of FGO's optimization pipeline. The system operates in three stages: (1) Divide: the full dataset is split into manageable subsets, (2) Optimize: parallel optimization is performed on each subset using LLM-based optimization with multiple iterations, and (3) Merge: optimized modules 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, their tasks, and performances
Output: Optimized module θ^*
function PROGRESSIVEMERGE($\mathcal{M}$, t)
  if $|\mathcal{M}| \leq t$ **then**
    $\theta, p \leftarrow \text{Merge}(\mathcal{M})$ ▷ Base: Direct Merge
    return θ
  end if
  $C \leftarrow \text{KMeans}(S, \sqrt{|\mathcal{M}|})$ ▷ Adaptive cluster
  for each cluster $c_i \in C$ **do**
    $\theta_i, p_i \leftarrow \text{ProgressiveMerge}(c_i, t)$
  end for
  return $\text{Merge}(\{\theta_i, p_i\})$
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 the immediate scalability constraints, it introduces the challenge of integrating N independently optimized modules while preserving their specialized insights. The straightforward approach would be to directly prompt an LLM with all module-performance pairs and generate an updated module. However, such all-at-once merging struggles to effectively process and synthesize patterns across many modules simultaneously, potentially losing the specialized optimizations gained through divided optimization. We propose progressive merging, implemented as a recursive algorithm that controls merging granularity through a cluster size threshold. Algorithm 2 shows the process of progressive merging. For a given list of module-performance pairs, we first check if the list size exceeds the threshold. For larger lists, we partition it into $k = \lfloor\sqrt{n}\rfloor$ clusters based on similarities, where n is the number of modules. Each resulting cluster then undergoes recursive merging. When a cluster's size falls below the threshold, we merge its modules by prompting an LLM with the module contents and their corresponding performance statistics. After each merge operation, we evaluate the merged module's performance through validating on the combined task set from all constituent modules. The recursive process naturally builds 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 stays within LLM context limits while capturing intricate relationships between similar modules, ultimately enabling efficient optimization of large-scale agentic systems.

4 Evaluations

4.1 Experiment Setup

We evaluate FGO by optimizing two different modules of the agentic system: instructions and tools. With proper instructions on the guidelines for the tasks, the agent can comprehend the scenario and solve it with ease (Fu et al., 2024; Chen et al., 2024; Wu et al., 2024b; Zhao et al., 2024). The tools expand the action space available to the agent, functioning as specialized modules that enable specific capabilities. Optimizing the tool configuration directly impacts the agent's ability to execute complex tasks efficiently and accurately.

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

- **ALFWorld** (Shridhar et al., 2020) is a text-based benchmark in which the agent is tasked with performing household tasks. Given a high-level objective, the agent needs to interact with the virtual environment and perform actions through natural language to finish the task. We randomly select 60 tasks from the training datasets (10 for each task type), and use the unseen set containing 134 tasks as test set. The benchmark contains 6 types of tasks, we set the number of agent optimizers to 6, with each agent optimizer optimizing each type of task. We report the success rate on different types of tasks and the overall success rate.
- **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. The dataset contains 267 document instances. For further details of the dataset, please refer to Appendix B. We randomly select 48 documents for training, the remaining 219 for testing. We set the number of agent optimizers to 8, each performs optimization on randomly split 6 tasks. A task is considered successful if the agent's answer is an exact match with the ground truth.
- **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. We utilize 36 tasks from the training set and evaluate on 60 tasks. The optimization is distributed across 4 optimizers, each handling a distinct subset of tasks.

Baselines for Comparison. We compare performance with different agent optimization methods: (1) **All-at-once** optimization represents the conventional approach of performing agent optimization on the whole training set using the algorithm illustrated in Section 3.2; (2) **Batch-wise** optimization employs a fixed-size batching strategy, splitting the training dataset into predetermined chunks and performing optimization sequentially on each task batch within an epoch; (3) **Bootstrapping** optimization implements a stochastic approach, sampling task batches from the training dataset with replacement.

Implementation details. We optimize the instructions for the agent on ALFWorld and LogisticsQA, and optimize the tools on GAIA. For ALFWorld, we leverage gpt-4o-mini as the backbone for the agent and evaluator, and gpt-4o for optimization and merging. For LogisticsQA and GAIA, we use gpt-4o in the whole process. For a fair comparison, all methods use the same number of optimization steps.

4.2 Main Results

Finding 1: FGO demonstrates superior optimization performance across multiple domains. We present the optimized agent's performance on different bencmarks in Table 1. For the majority of

Methods	ALFWorld							LogisticQA	GAIA
	Pick	Clean	Heat	Cool	Look	Pick Two	Average		
Vanilla Agent	69.4	50.5	65.2	20.6	31.5	21.6	45.5	36.3	15.0
All-at-once	90.2	72.6	78.3	78.6	66.7	55.9	75.0	52.1*	21.7*
Batch-wise	77.1	71.0	67.4	64.3	**86.1**	**73.5**	72.8	55.7	10.0
Bootstrapping	**91.7**	77.4	73.9	74.6	87.0	41.2	75.6	62.6	20.0
FGO	90.2	**83.8**	**87.0**	**88.9**	**86.1**	62.7	**83.6**	**64.8**	**23.3**

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

the tasks, the agent demonstrates performance gain in most cases after optimization. This highlights how targeted prompt and tool refinement can significantly enhance LLM agent capabilities. Among the optimization methods, FGO achieves the most performance boost in all cases, with gains ranging from 8.3% to 38.1% compared to the vanilla agents.

Finding 2: Progressive merging effectively preserves task-specific patterns while achieving global optimal. The superior performance of FGO can be attributed to its divide-and-conquer-based methodology. The All-at-once approach processes the entire training dataset simultaneously, requiring the LLM optimizer to learn from trajectories across the complete dataset. This leads to suboptimal performance due to the optimizer's difficulty in processing complex patterns in long corpus, as evidenced by the suboptimal performance on ALFWorld subtasks. Alternative methods like bootstrapping optimization and batch-wise optimization demonstrate strong performance in specific categories, but fail to maintain consistent performance across the task spectrum. Their batch-wise optimization approach introduces instability in the training process, as the LLM optimizer encounters different data distributions in successive iterations, potentially compromising previously learned patterns. In contrast, FGO overcomes these limitations through its systematic merging of independently optimized instructions and tools. By first optimizing subset-specific instructions and tools and then progressively merging them, FGO can preserve task-specific patterns while building toward global optimization. We further examine the implication of merging on FGO performance in Section 4.4.

4.3 Further Analysis

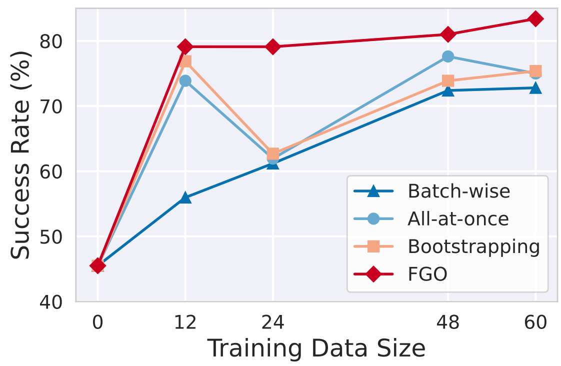

Figure 3: Analysis of the number of training tasks. We run optimization on varied training dataset sizes and plot the performance. FGO achieves best performance in all training settings, and demonstrate a steady increase as the training taskset size increases.

Finding 3: FGO demonstrates extraordinary scalability. We evaluated how training data volume affects optimization performance on ALFWorld. As shown in Figure 3, FGO maintains stable performance across all dataset sizes, with consistent improvements as training samples increase. 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 learning principles where bootstrapping enhances generalization (Breiman, 1996). Additionally, All-at-once optimization proves impractical for LogisticsQA and GAIA due to their extensive document lengths (>3,000 tokens) and complex solution trajectories exceeding LLM context windows, validating the need for our scalable approach.

Finding 4: FGO achieves an optimal balance between token cost, efficiency and performance. We visualize the relationship between prompt to-

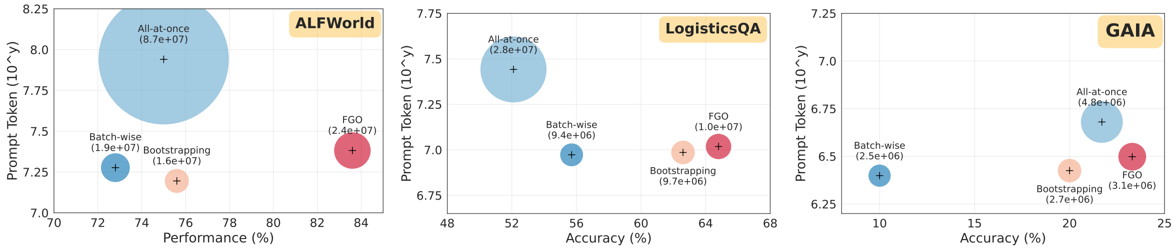

Figure 4: Comparison of prompt token efficiency across different optimization methods on ALFWorld, LogisticsQA, and GAIA. Each panel plots the trained agent's performance against the total prompt tokens consumed during optimization. Circle diameters are proportional to the optimization token consumption, with crosses (+) indicating circle centers.

ken used for optimization and the performance after training with different optimization methods in Figure 4, and outline the time to train and performance in Figure 5. In terms of token cost, FGO requires larger number of prompt tokens compared to Batch-wise optimization and Boostrapping 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 contrast, All-at-once prompts the LLM optimizer with the whole list of trajectories and losses, leading to the largest token consumption requirements than other methods. 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 components and hyperparameters in FGO:

How does progressive merging and choice of clustering algorithm affect performance? To analyze the impact of progressive merging and evaluate different clustering algorithms in the merging process, we conduct experiments on the ALFWorld benchmark. We first establish a baseline by removing the progressive merging entirely, and instead prompting an LLM to generate the final module directly from the module-performance pairs obtained from divided optimization. We then evaluate the effect of different clustering algorithms by fixing the independently optimized modules and changing the clustering method to Spectral clustering and Bisect K-Means. We report the average and best of three runs in Table 2. Without progressive merging, the method achieves a 73.1% average success rate, demonstrating that even basic merging provides substantial improvement over the vanilla agent. In comparison, the introduction of progressive merging significantly boosts performance. Regardless of the clustering algorithm employed during merging, the final performances all demonstrate consistent improvement to no merging. This consistency suggests that the progressive nature of the merging strategy, rather than the specific clustering algorithm, is the key driver of improvement.

Methods	Cluster Algorithm	ALFWorld	
		Avg of 3	Best of 3
Vanilla	-	45.5	61.9
FGO	None	73.1	84.3
	Spectral	81.6	89.6
	Bisect K-Means	80.1	**91.0**
	K-Means	**83.6**	89.6

Table 2: Ablation study on the effects of clustering algorithms used. "None" means we skip the clustering step and directly merge the optimized modules.

Does the division of training data affect performance? To examine the robustness of FGO, we compare our default category-based partitioning against random partitioning of training tasks in ALFWorld. As shown in Table 3, while random partitioning shows a slight performance drop, the system still maintains strong performance thanks to the progressive merging process, which effectively combines optimization insights across partitions. This demonstrates that FGO's performance remains robust even with suboptimal partitioning strategies.

How does the number of divided subsets affect performance? To answer this question, we conduct ablation study on the number of independent

Partition	ALFWorld	
	Avg of 3	Best of 3
Random	80.3	88.1
Category	**83.6**	**89.6**

Table 3: Ablation on the data partitioning strategy. 'Category' denotes we partition the training data according to the task type.

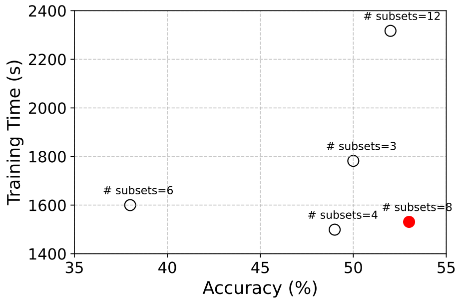

Figure 5: Ablation study on the number of divided subsets. Most parameter settings achieve similar performance, with varying time for optimization.

agent optimizers. We trained agents on LogisticsQA and set the number of divided subsets to 3, 4, 6, 8, 12. Due to the high cost in running `gpt-4o`, we sampled 100 tasks from the test set and validate the optimized agent's performance respectively. We plot the relationship between performance and training time in Figure 5. The results suggest that the choice of agent number primarily impacts computational efficiency rather than optimization quality.

How does performance change with respect to backbone LLM? Finally, to ensure FGO well generalizes to different backbones, we change the optimizer backbone to o3-mini and observe the metrics. We report the token consumption for training, wall-clock time, and performance in Table 4. In line with the main results, FGO maintains strong performance while reaching most efficiency, with slightly overhead in token consumption.

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

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

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; Opsahl-Ong et al., 2024; 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 a meta-optimizers to improve an LLM based system (Zelikman et al., 2023; Yin et al., 2024).

Automated Agentic System Design. There has been efforts in exploring inference time performance boost since the emergence of Large Language Models (Shinn et al., 2024; Madaan et al., 2024; Yao et al., 2023, 2024; Wei et al., 2022; Guo et al., 2024). Recent works have extended this paradigm to agentic systems. Some works represent and learn the optimal workflow of agentic systems in the form of complex graphs (Zhuge et al., 2024; Wu et al., 2024c), code (Hu et al., 2024a), and trees (Zhang et al., 2024a) to improve the system's performance on complex tasks, while others learns reusable tools (Zhang et al., 2024c; Cai et al., 2023; Qian et al., 2023; Yuan et al., 2023) and experience (Zhao et al., 2024; Wang et al., 2024b) for agentic systems.

6 Conclusion

In this paper, we addressed the scalability challenges in LLM-based agent optimization by introducing FGO, a framework that effectively processes large-scale execution trajectories through task division, fine-grained optimization, and progressive module merging. Our evaluation across multiple dataset demonstrates consistent performance improvements. FGO reaches an optimal balance between performance, efficiency and token consumption.

Limitations

The merging process introduces computational overhead, as it requires to back test the merged module on the merged training dataset, resulting in larger token cost compared to Batch-wise optimization and Bootstrappingoptimization. In future works, 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. *arXiv preprint arXiv:2406.16218*.

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*.

Xudong Guo, Kaixuan Huang, Jiale Liu, Wenhui Fan, Natalia Vélez, Qingyun Wu, Huazheng Wang, Thomas L Griffiths, and Mengdi Wang. 2024. Embodied llm agents learn to cooperate in organized teams. *arXiv preprint arXiv:2403.12482*.

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*.

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*.

Mingyang Song, Mao Zheng, and Xuan Luo. 2024. Counting-stars: A simple, efficient, and reasonable strategy for evaluating long-context large language models. *arXiv preprint arXiv:2403.11802*.

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*.

Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. 2024c. Stateflow: Enhancing llm task-solving through state-driven workflows. *arXiv preprint arXiv:2403.11322*.

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*.

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, 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*.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Hao, Xu Han, Zhen Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2024d. ∞Bench: Extending long context evaluation beyond 100K tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15262–15277, Bangkok, Thailand. Association for Computational Linguistics.

Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. 2024e. 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

How is the efficiency of FGO in terms of optimizing time consumption? In this section, we present the time for optimization on each method. Ours are the most efficient thanks to the parallel implementation.

How does the trained agent perform after different optimization methods? We plot the token consumption for inferencing on the dataset with the modules trained with different methods. As shown in Table 6, 7, 8,FGO can reach competent performance with reasonable token consumption overhead.

B LogisticQA Dataset

B.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.

Methods	ALFWorld	LogisticQA	GAIA
all	8372	3600	7400
batch	2975.67	2550	4798
bootstrap	2567.72	2621	3957
FGO	1998	2434.0691	5906

Table 5: Performance comparison across different methods on ALFWorld, LogisticQA and GAIA datasets.

Method	Tokens	Performance
All-at-once	8,856,145	75.0
Batch-wise	9,018,478	72.8
Boostrapping	8,172,052	75.6
FGO	**7,594,598**	**83.6**

Table 6: Inference cost and performance for ALFWorld.

Method	Tokens	Performance
All-at-once	6,483,562	52.1
Batch-wise	6,872,656	55.7
Boostrapping	**5,703,318**	62.6
FGO	6,287,424	**64.8**

Table 7: Inference cost and performance for LogisticsQA.

B.2 Dataset Statistics

The analysis of our XML business document dataset demonstrates strong alignment with real-world business documentation patterns, as shown in Figure 6. 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.

B.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
<?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>
```

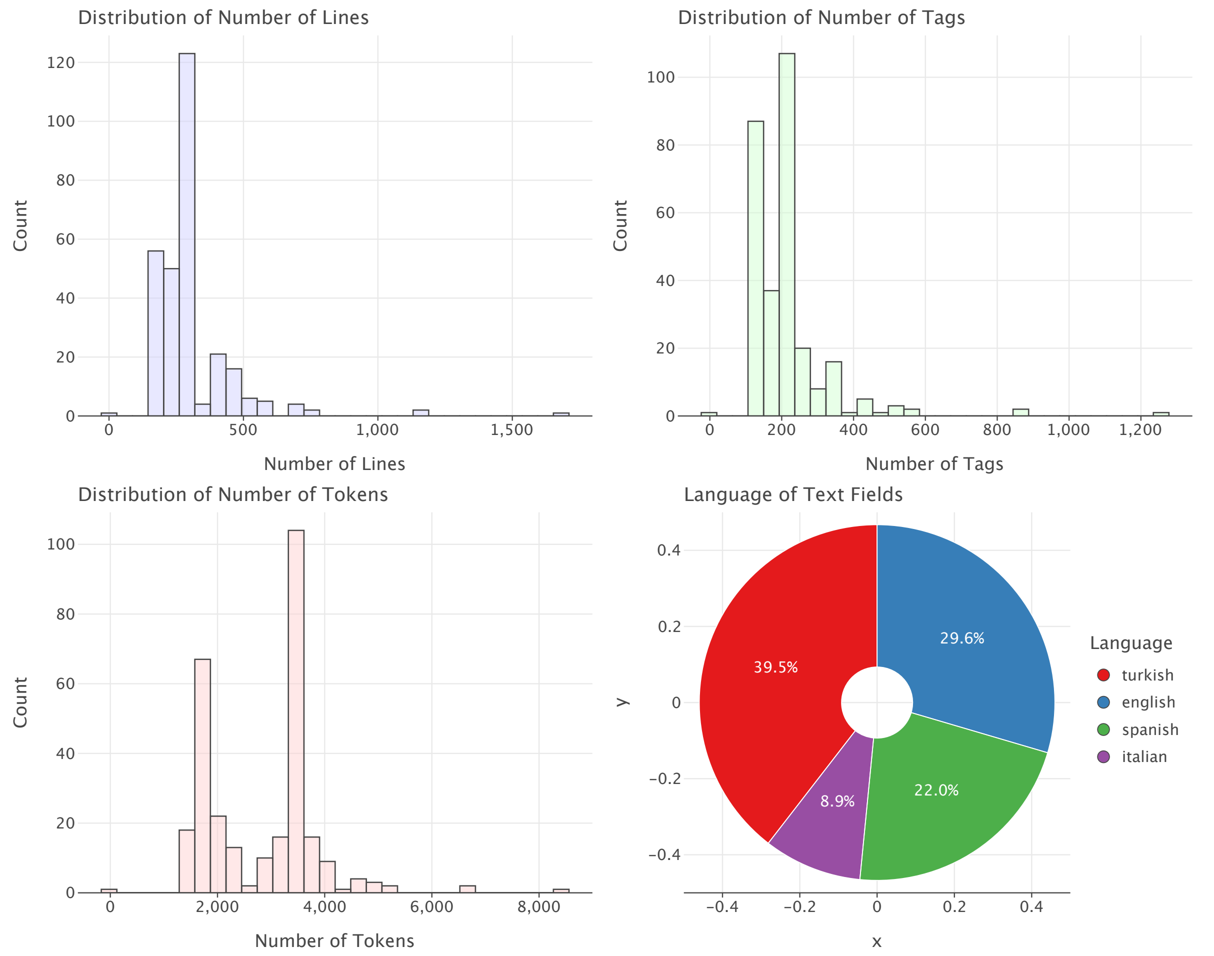

Figure 6: Statistical analysis of XML business documents. Top left: Distribution of document lengths showing typical business document sizes. Top right: Distribution of XML tags indicating document structure complexity. Bottom left: Token distribution demonstrating the long context challenge for LLM. Bottom right: Language distribution across documents reflects business documents' multinational nature.

Method	Tokens	Performance
All-at-once	**527,551**	21.7
Batch-wise	436,337	10.0
Boostrapping	877,283	20.0
FGO	787,921	**23.3**

Table 8: Inference cost and performance for GAIA.


```
    <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).
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>
```


C Complexity Analysis

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

C.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:

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

C.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.

D Prompt

D.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.
```

D.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]"
```

D.3 GAIA

```
# Task
You need to solve the question below
   given by a user. When you are
   building tasks, explicitly consider
   where the task can benefit from web
   navigation capability.

# Task
{task}
"""
```