GBC: Gradient-Based Connections for Optimizing Multi-Agent Systems

Xiaocheng Yang, Abdulrahman Alrabah, Dilek Hakkani-Tür, Gokhan Tur

University of Illinois Urbana-Champaign

{xy61, alrabah2, dilek, gokhan}@illinois.edu

Abstract

Multi-agent systems (MAS) built on large language models (LLMs) provide a promising framework for solving complex tasks through role specialization and structured interaction. However, their performance is often limited by miscoordination and, more fundamentally, the lack of fine-grained credit assignment across agents. Existing approaches typically rely on coarse-grained feedback, making it difficult to identify which agents or interaction steps are responsible for errors. We propose Gradient-Based Connections (GBC), an approach for fine-grained attribution and optimization of multi-agent systems. GBC models a MAS as a computational graph and introduces gradient-based connection weights to quantify the influence of each agent's output on downstream agents at the token level. By constructing an attribution graph and propagating task-specific loss signals backward, our method enables precise identification of error sources and targeted prompt optimization. We further develop AgentChord, an efficient implementation that leverages prefix-based gradient computation. Experiments on MultiWOZ and τ-bench show that GBC improves multi-agent performance and outperforms strong single-agent and multi-agent baselines, and higher attribution quality is associated with greater optimization effectiveness. Code is available at: https://github.com/yxc-cyber/AgentChord.

1 Introduction

Large Language Models (LLMs) have enabled a new paradigm of multi-agent systems (MAS), where multiple specialized agents collaborate through structured interactions to solve complex tasks. By decomposing problems into sub-tasks and assigning them to different agents, multi-agent systems have demonstrated promise across a wide range of domains, including task-oriented dialogue, software engineering, and open-ended simulations (Gupta et al., 2024; Wu et al., 2023; Qian et al., 2024). However, despite their conceptual appeal, recent studies show that multi-agent systems often fail to consistently outperform strong single-agent baselines and suffer from issues such as miscoordination, inefficient communication, and lack of robust verification (Pan et al., 2025).

A fundamental challenge underlying these limitations is the lack of fine-grained credit assignment. In multi-agent workflows, errors in the final output often originate from specific agents or interaction steps, yet existing methods typically rely on coarse-grained signals (e.g., overall task success or reward) to guide optimization (Khattab et al., 2024; Xu et al., 2025; Yuksekgonul et al., 2024; Zhuge et al., 2024; Luo et al., 2025). This makes it difficult to identify which components of the system are responsible for failures, limiting the effectiveness of both manual debugging and automatic optimization.

Recent advances in prompt optimization and LLM-based system design have explored gradient-inspired methods for improving performance, such as textual feedback propagation and self-supervised optimization (Yang et al., 2024; Zhou et al., 2023; Pryzant et al., 2023; Xiang et al., 2025). While these approaches introduce more structured optimization signals, they are primarily designed for single-agent or monolithic pipelines, and do not explicitly address the challenges of attribution and optimization in multi-agent settings. In parallel, prior work on multi-agent systems has focused on improving coordination through architectural design—such as role specialization, graph-based communication, and task decomposition—but largely lacks principled mechanisms for token-level or interaction-level attribution across agents.

To address these challenges, we propose Gradient-Based Connections (GBC), a novel approach for optimizing multi-agent systems through fine-grained attribution. We model a multi-agent system as a directed computational graph and in-

troduce a gradient-based connection mechanism that quantifies the influence of each agent's output on subsequent agents at the token level. By constructing an attribution graph over agent interactions and propagating task-specific verbal loss signals backward through this graph, GBC enables precise identification of the components most responsible for errors. This facilitates more effective and targeted optimization of agent prompts.

Building on this formulation, we develop AgentChord, a practical framework that integrates GBC with a language-model-based optimizer to iteratively refine multi-agent systems. To ensure scalability, we introduce an efficient implementation that leverages a prefix-based gradient computation strategy to reduce memory overhead during backpropagation.

We evaluate our approach on both task-oriented dialogue (MultiWOZ (Ye et al., 2022)) and interactive tool-use environments (τ-bench (Yao et al., 2025)), demonstrating that GBC significantly improves multi-agent performance across multiple metrics. Our results show that fine-grained attribution enables effective optimization, and in some cases allows multi-agent systems to surpass strong single-agent baselines. Analysis of the MultiWOZ results further reveals an association between attribution quality and optimization effectiveness.

Our contributions are summarized as follows:

- We propose Gradient-Based Connections (GBC), a method for token-level attribution across agents via gradient-based signals.
- We introduce AgentChord, a scalable framework for optimizing multi-agent systems using attribution-guided updates.
- We demonstrate the effectiveness of our approach on multiple benchmarks, including MultiWOZ and τ-bench, showing improvements over single-agent and multi-agent baselines.

2 Related Work

2.1 Prompt Optimization

The performance of large language models (LLMs) is highly sensitive to prompt design, motivating extensive work on automatic prompt optimization. Early approaches formulate prompt design as black-box search over natural language instructions, where candidate prompts are generated and evaluated iteratively using LLMs themselves (Zhou et al., 2023; Yang et al., 2024). These methods leverage LLMs as both generators and evaluators but rely on exploration guided by coarse-grained performance signals.

A complementary line of work introduces gradient-inspired optimization for prompts. ProTeGi models prompt refinement as following "textual gradients", where natural language feedback describing model errors is used to iteratively update prompts (Pryzant et al., 2023). More generally, TextGrad extends this idea by treating LLM-based systems as computation graphs and propagating feedback signals across components, enabling automatic differentiation over prompt variables (Yuksekgonul et al., 2024). Recent work further explores self-supervised optimization. Self-Supervised Prompt Optimization (SPO) eliminates the need for labeled data by deriving optimization signals from pairwise comparisons of model outputs (Xiang et al., 2025).

In addition, evolutionary approaches such as GAAPO and GEPA apply genetic algorithms to explore diverse prompt candidates through mutation and selection (Sécheresse et al., 2025; Agrawal et al., 2026).

Despite these advances, these methods primarily focus on optimizing prompts for single-agent or single-step generation settings and rely on global performance feedback. They lack mechanisms for fine-grained attribution of errors to specific tokens, intermediate reasoning steps, or interacting components, limiting their effectiveness in complex multi-agent systems.

2.2 Multi-Agent System

Multi-agent systems (MAS) built on large language models (LLMs) have emerged as a powerful paradigm for solving complex tasks via role specialization, task decomposition, and iterative inter-agent communication. General-purpose frameworks such as AutoGen and ChatDev demonstrate that flexible multi-agent conversations and structured role-based collaboration can effectively coordinate LLM agents for complex workflows (Wu et al., 2023; Qian et al., 2024). MAS have since been applied across diverse domains, including research simulation, open-ended social environments, and task-oriented dialogues (Yu et al., 2025; Ye and Jaques, 2024; Gupta et al., 2024).

To improve scalability and coordination, recent work introduces structured architectures for orga-

nizing agent interactions. Graph-based formulations represent agents and their communications as computational graphs, enabling systematic reasoning and optimization of information flow (Zhuge et al., 2024). Similarly, DAG-based collaboration networks and task dependency graphs structure agent interactions for scalable coordination and complex task execution (Qian et al., 2025; Dong et al., 2024). Complementary approaches improve efficiency via role-aware routing and dynamic context selection (Liu et al., 2025).

Despite these advances, recent studies show that MAS often provide limited gains over strong single-agent baselines and suffer from inter-agent misalignment, inefficient coordination, and weak verification (Pan et al., 2025). Moreover, identifying the source of errors remains challenging, as failures arise from specific agents or interaction steps but are difficult to attribute automatically (Zhang et al., 2025a). These limitations are further compounded by broader risks such as miscoordination and emergent behaviors in complex multi-agent settings (Hammond et al., 2025).

2.3 Multi-Agent System Optimization

Beyond static architectures, a growing line of work treats multi-agent systems as optimizable programs. DSPy compiles declarative LLM pipelines into optimized execution graphs (Khattab et al., 2024), while TextGrad and metaTextGrad introduce gradient-like optimization mechanisms that propagate natural language feedback through computation graphs (Yuksekgonul et al., 2024; Xu et al., 2025). In the multi-agent setting, GPTSwarm optimizes both agent behaviors and inter-agent connectivity within graph-structured systems, while MetaAgent automates the construction of agent organizations (Zhuge et al., 2024; Zhang et al., 2025b). Reinforcement learning approaches further enable system-level optimization by learning from interaction trajectories and addressing credit assignment across agents (Luo et al., 2025).

Overall, existing approaches either rely on manually designed coordination mechanisms or optimize systems using coarse-grained signals, without providing fine-grained attribution across agents and interaction steps. In contrast, our work introduces a gradient-based connection framework that enables token-level attribution across agents, allowing more precise credit assignment and principled optimization of multi-agent systems.

3 Method

We propose a framework for optimizing multi-agent systems with four components: agent graph, gradient-based connection, loss, and optimizer. Figure 1 illustrates the overall pipeline. Given an input, the agent graph produces a final output through sequential agent interactions. Gradient-based connections construct an attribution graph that quantifies the influence of each predecessor output. A task-specific verbal loss is attached to the final output, and gradients are propagated backward to extract attribution trajectories identifying error sources. The optimizer then updates agent prompts based on these trajectories.

3.1 Agent Graph

We model the forward procedure of a multi-agent system as a directed acyclic graph $G = (V, E)$, following Zhuge et al. (2024). Each agent $v \in V$ is defined by a prompt–model pair (P_v, M_v). Edges $E = \{(v_1, v_2) \mid O_{v_1} \subseteq I_{v_2}\}$ represent information flow.

The output of agent v is:

$$O_v = M_v(P_v + I_v). \tag{1}$$

The input is:

$$I_v = \begin{cases} I_{\text{initial}}, & \text{if } v \text{ is the first node,} \\ \sum_{u \in \text{pre}(v)} O_u, & \text{otherwise.} \end{cases} \tag{2}$$

Agents are executed in topological order, and the final output is produced by the last agent.

3.2 Gradient-Based Connection

We introduce gradient-based connections to quantify the contribution of each predecessor output. For each O_v, we compute connection weights $W_{O_v}(O_u)$ for $u \in \text{pre}(v)$ using gradient-based signals. We consider four variants: mean/max of L1 norm and mean/max of gradient–input product (Sections 3.2.1–3.2.4).

For each output, we retain the top-m predecessors to construct an attribution graph $G_{\text{attr}} = (V_{\text{attr}}, E_{\text{attr}})$:

$$E_{\text{attr}} = \{(v_{\text{attr},1}, v_{\text{attr},2}) \mid (v_{\text{attr},1}, v_{\text{attr},2}) \in V_{\text{attr}}^2, \\ W_{O_{v_2}}(O_{v_1}) \in \text{Top}_m(W_{O_{v_2}})\}. \tag{3}$$

We use $m = 1$ by default.

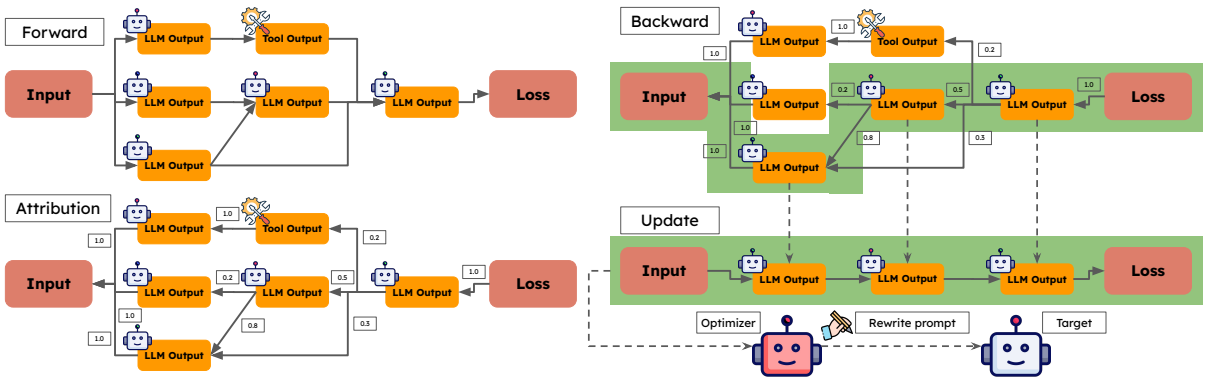

Figure 1: Overview of multi-agent system optimization with GBC. The procedure consists of four steps: **(1) Forward**, **(2) Attribution**, **(3) Backward**, and **(4) Update**. **(1) Forward**: The agent graph processes the input sequentially and produces the final output. **(2) Attribution**: Gradient-based connections construct an attribution graph that quantifies the influence of each predecessor output. **(3) Backward**: The framework propagates the loss backward through the attribution graph to extract attribution trajectories that identify the outputs most responsible for the final result. **(4) Update**: The optimizer updates agent prompts based on these trajectories to improve overall performance.

3.2.1 Mean of L1 Norm

$$W_{O_v}(O_u) = \text{avg}\Big(\big\|\nabla \prod_{w \in O_v} \mathbb{P}(w \mid \text{Embed}(O_u))\big\|_{L_1}\Big). \tag{4}$$

This computes token-level salience scores via gradients and aggregates them by averaging.

3.2.2 Max of L1 Norm

$$W_{O_v}(O_u) = \max\Big(\big\|\nabla \prod_{w \in O_v} \mathbb{P}(w \mid \text{Embed}(O_u))\big\|_{L_1}\Big). \tag{5}$$

This emphasizes the most influential tokens, reducing noise from irrelevant ones.

3.2.3 Mean of Product with Input

$$W_{O_v}(O_u) = \text{avg}\Big(\nabla \prod_{w \in O_v} \mathbb{P}(w \mid \text{Embed}(O_u)) \cdot \text{Embed}(O_u)\Big). \tag{6}$$

This captures first-order contributions via gradient–input interactions (Shrikumar et al., 2017).

3.2.4 Max of Product with Input

$$W_{O_v}(O_u) = \max\Big(\nabla \prod_{w \in O_v} \mathbb{P}(w \mid \text{Embed}(O_u)) \cdot \text{Embed}(O_u)\Big). \tag{7}$$

This combines the gradient–input interactions with the emphasis on most influential tokens.

3.3 Loss

We define a task-specific verbal loss based on the system output and attach it to the attribution graph. The loss encodes correctness and quality signals, and can include fine-grained feedback (e.g., ground truth comparisons or explanations) to guide optimization. Details are provided in Appendix B.

3.4 Optimizer

We backpropagate the loss through the attribution graph to obtain attribution trajectories:

$$\tau = [(s_0, c_0), \ldots, (\ell, L_\ell)].$$

Each trajectory traces the contribution from inputs or intermediate outputs to the loss. The set $\mathcal{T}(\text{input})$ collects all such trajectories. The procedure is summarized in Algorithm 1.

We then use a language model as the optimizer (Yang et al., 2024). Given current prompts, attribution trajectories, and optimization history, it updates agent prompts to improve performance. Details are provided in Appendix C.

4 AgentChord

Following the pipeline in Section 3, we develop **AgentChord**[1], a practical framework for multi-agent prompt optimization using Gradient-Based Connections (GBC).

To enable scalability, we introduce a *prefix-based gradient computation* technique to reduce memory overhead. As defined in Equation 1, each agent processes a prompt and an input. Since attribution is computed only with respect to the input, gradients are required only for input tokens, while prompt tokens are treated as a fixed prefix.

[1]Code is available at: https://github.com/yxc-cyber/AgentChord.

Algorithm 1 Backpropagation of Attribution Trajectories

Require: Attribution graph $G_{\text{attr}} = (V_{\text{attr}}, E_{\text{attr}})$
Require: Loss node set $V_{\text{loss}} = \{(\ell, L_\ell)\}$
Require: Initial input I_{initial}
Initialize an empty list $\mathcal{T}(s)$ for each subject $s \in V \cup \{I_{\text{initial}}\}$
for all $(\ell, L_\ell) \in V_{\text{loss}}$ **do**
  cache $\leftarrow [[(\ell, L_\ell)]]$
  for all $(v, O_v) \in \text{pre}_{\text{attr}}(\ell, L_\ell)$ **do**
    BACKWARD$((v, O_v), \text{cache})$
  end for
end for
return $\{\mathcal{T}(s)\}_{s \in V \cup \{I_{\text{initial}}\}}$
function BACKWARD$((v, O_v), \text{cache})$
  new_cache $\leftarrow [\,]$
  for all $\tau \in$ cache **do**
    $\tau' \leftarrow \text{copy}(\tau)$
    Insert (v, O_v) at the beginning of τ'
    Append τ' to new_cache
  end for
  $\mathcal{T}(v) \leftarrow \mathcal{T}(v) \cup$ new_cache
  if $\text{pre}_{\text{attr}}(v, O_v) = \emptyset$ **then**
    input_cache $\leftarrow [\,]$
    for all $\tau \in$ new_cache **do**
      $\tau' \leftarrow \text{copy}(\tau)$
      Insert $(input, I_{\text{initial}})$ at the beginning of τ'
      Append τ' to input_cache
    end for
    $\mathcal{T}(input) \leftarrow \mathcal{T}(input) \cup$ input_cache
  else
    for all $(u, O_u) \in \text{pre}_{\text{attr}}(v, O_v)$ **do**
      BACKWARD$((u, O_u), \text{new_cache})$
    end for
  end if
end function

In practice, we first pass the prompt through the model without gradients to obtain the KV cache, and then process the input with gradients enabled. This avoids storing gradients for prompt tokens.

As a result, memory complexity is reduced from

$$\mathcal{O}(n \cdot d \cdot L)$$

to

$$\mathcal{O}((n - k) \cdot d \cdot L),$$

where n is the total sequence length, k is the prompt length, d is the hidden dimension, and L is the number of layers.

5 Experiment

Multi-agent systems enable task decomposition and specialization, which can improve performance in complex, multi-domain settings. We evaluate GBC on two benchmarks: MultiWOZ for task-oriented dialogue and τ-bench for interactive tool-use.

5.1 MultiWOZ

Setup MultiWOZ (Ye et al., 2022; Budzianowski et al., 2018; Eric et al., 2020; Zang et al., 2020; Han et al., 2021) is a task-oriented dialogue benchmark with annotated dialogue states. We use MultiWOZ 2.4 and sample 100 dialogues from five domains (Attraction, Hotel, Restaurant, Train, Taxi).

We adopt a manager–worker architecture (Figure 2). The manager assigns tasks based on dialogue context, domain-specific workers perform API calls, and a responder generates the final response.

Verbal Loss We use two types of verbal loss: (1) a turn-level JGA loss that compares predicted and ground-truth dialogue states, including false positive and false negative slot-value pairs; and (2) a dialogue-level Inform & Success loss that evaluates whether the system retrieves correct entities and provides requested information. Detailed prompts are provided in Appendix B.1.

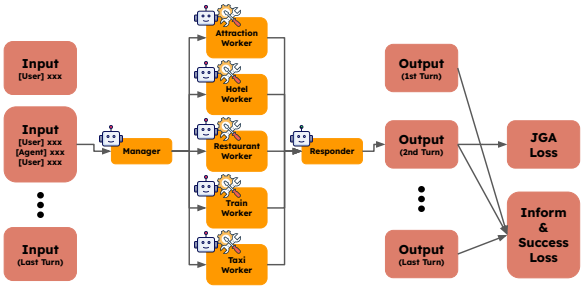

Figure 2: Multi-agent system tailored to MultiWOZ.

Model	Inform	Success	JGA	Recall	Precision	Slot F1
Single-Agent						
Llama-3.3-70B-It	84.0	71.0	40.3	96.9	81.5	88.5
Qwen-3-32B	88.0	40.0	44.4	97.7	80.9	88.5
Multi-Agent (Before Optimization)						
Llama-3.3-70B-It	87.0	57.0	24.3	97.4	54.1	69.5
Qwen-3-32B	95.0	80.0	28.9	92.5	69.4	79.3
Multi-Agent (Optimized with Mean of L1 Norm)						
Llama-3.3-70B-It	42.0	07.0	28.6	90.7	78.0	83.9
Qwen-3-32B	**99.0**	**94.0**	**54.4**	**98.1**	**85.5**	**91.4**
Multi-Agent (Optimized with Max of L1 Norm)						
Llama-3.3-70B-It	87.0	54.0	36.7	94.5	78.9	86.0
Qwen-3-32B	**99.0**	**95.0**	**53.0**	**98.3**	**85.2**	**91.3**
Multi-Agent (Optimized with Mean of Product with Input)						
Llama-3.3-70B-It	82.0	59.0	39.0	96.7	79.6	87.4
Qwen-3-32B	98.0	83.0	51.9	96.9	84.6	90.3
Multi-Agent (Optimized with Max of Product with Input)						
Llama-3.3-70B-It	85.0	62.0	38.9	94.7	81.7	87.7
Qwen-3-32B	95.0	86.0	51.0	97.4	79.3	87.4

Table 1: Single-agent System and multi-agent system performance on MultiWOZ 2.4. The table shows the inform score, success score, joint goal accuracy (JGA), slot recall, slot precision, and slot F1 score. **Best results** are bold and underlined; **second-best** are bold.

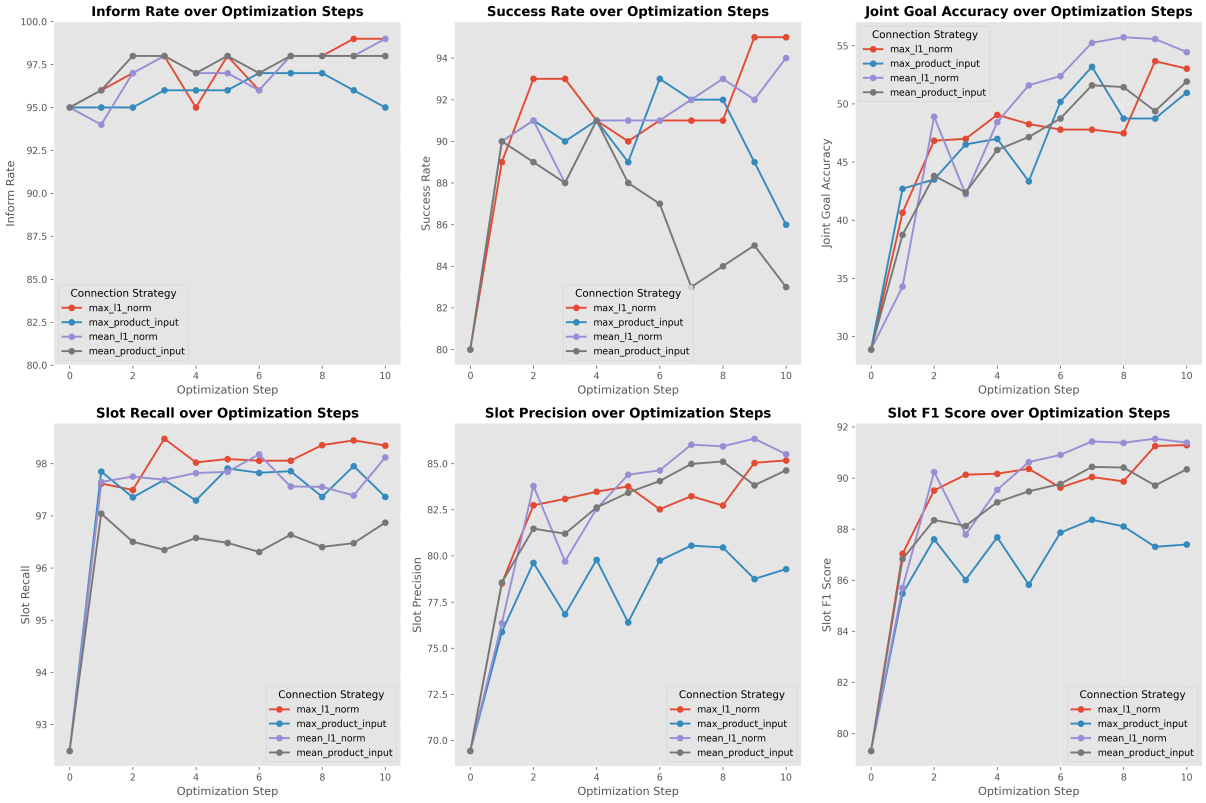

Figure 3: Optimization dynamics of Qwen-3-32B on MultiWOZ. Step 0 denotes the unoptimized multi-agent baseline. **Inform** and **Slot Recall** remain high throughout optimization, while **JGA**, **Slot Precision**, and **Slot F1** show clear upward trends, indicating improved dialogue-state tracking and fewer over-predicted slots. **Success** is more variable, suggesting that full goal completion remains harder to optimize. Overall, L1-norm-based connection weights achieve the strongest final performance.

Metrics We report Inform, Success (conversation-level), and Joint Goal Accuracy (JGA), Slot Recall, Slot Precision, and Slot F1 (turn/slot-level).

Optimization Setup We use 30 training samples, updating prompts every 3 samples (10 steps total). Backbone models are Llama-3.3-70B-It and Qwen-3-32B.

Results Results are shown in Table 1. Before optimization, the multi-agent systems do not consistently outperform their single-agent counterparts. For example, although the Qwen-3-32B multi-agent system achieves higher Inform and Success scores than the single-agent baseline, its JGA and Slot F1 are substantially lower. After optimization with GBC, performance improves across most metrics, especially for Qwen-3-32B. With mean of L1 norm, Qwen-3-32B reaches the best overall MultiWOZ performance, improving JGA from 28.9 to 54.4 and Slot F1 from 79.3 to 91.4, while also achieving 99.0 Inform and 94.0 Success. The max of L1 norm variant obtains similarly strong results, with 99.0 Inform, 95.0 Success, 53.0 JGA, and 91.3 Slot F1. These optimized multi-agent systems substantially outperform the Qwen-3-32B single-agent baseline on Inform, Success, JGA, and Slot F1, demonstrating that GBC can convert an initially under-optimized multi-agent system into a stronger task-oriented dialogue agent.

Figure 3 further illustrates the optimization dynamics for Qwen-3-32B. Across the four connection weight formulations, most metrics, including Inform, JGA, Slot Recall, Slot Precision, and Slot F1, show clear upward trends over optimization steps. In particular, JGA and Slot F1 improve steadily from the early steps, indicating that attribution-guided prompt updates help the system better track dialogue states and predict slot values. Success remains more variable and challenging, suggesting that completing the full user goal still depends on difficult long-horizon coordination. Overall, the trends in Figure 3 are consistent with Table 1 and show that GBC provides stable improvements during optimization, with the L1-norm-based variants yielding the strongest final

performance.

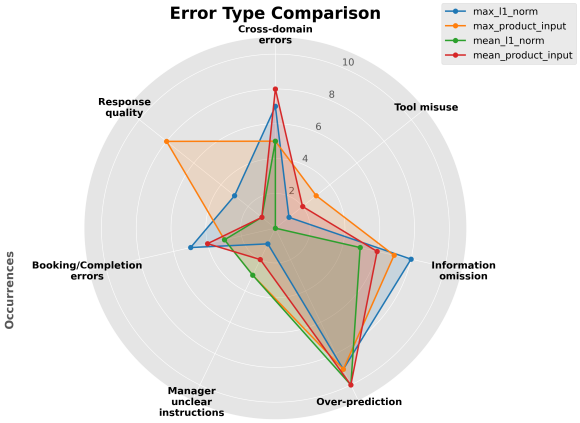

Figure 4: The occurrences of different error types detected by the optimizer under different connection weight formulae on MultiWOZ with Qwen-3-32B.

Error Analysis We categorize errors into seven types: cross-domain errors, tool misuse, information omission, over-prediction, unclear manager instructions, booking errors, and response quality issues.

As shown in Figure 4, cross-domain errors, information omission, and over-prediction occur most frequently. This suggests that MultiWOZ failures are mainly caused by multi-domain coordination and dialogue-state tracking, rather than only by surface-level response generation. Cross-domain errors indicate that the manager-worker routing is still imperfect, especially when multiple domain-specific agents are available. Information omission shows that even when relevant information appears in the dialogue, it may be lost during extraction or inter-agent communication. Over-prediction further suggests that agents sometimes infer slot values beyond the evidence provided by the user. These patterns are consistent with the improvements in JGA, Slot Precision, and Slot F1 after optimization, since reducing omitted and over-predicted slots directly improves dialogue-state tracking. Overall, the error distribution indicates that GBC improves the system by targeting coordination and state-tracking failures, while cross-domain responsibility assignment remains a key challenge.

Update Analysis Figure 5 shows normalized update frequency (NUF):

$$\mathrm{NUF}(v) = \frac{|\{i \mid v \in U_i\}|}{|\{i \mid v \in R_i\}|}. \tag{8}$$

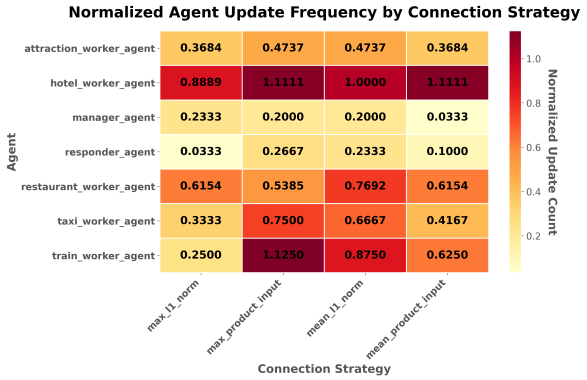

Figure 5: Heatmap of normalized agent update frequency by connection strategy with Qwen-3-32B.

U_i is the set of updated agents for the i-th round and R_i is the set of relevant agents for the i-th round. Responder and manager agents are always relevant, while domain-specific workers are relevant if and only if the task domain matches the work's domain.

Domain-specific workers are updated more frequently than manager or responder agents, consistent with the observed error patterns.

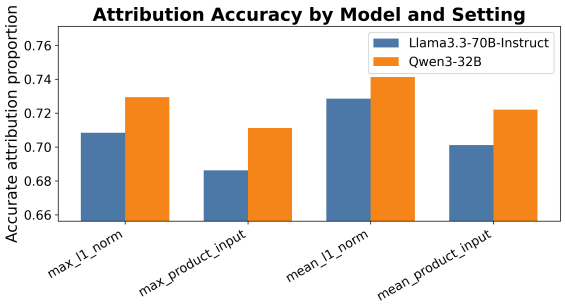

Figure 6: Attribution accuracy grouped by model within each connection weight setting.

Attribution Quality Analysis We approximate the accuracy of attribution by checking whether each attribution trajectory contains the worker agents responsible for the domains of the dialogue. Figure 6 shows the attribution accuracy for the 2 models and 4 connection weights. Regardless of the model, mean and max L1 norm always lead to the best two attribution accuracies, which explains why those two connection weight formulae perform best in terms of the metrics of the MultiWOZ task. This reveals that higher attribution quality is associated with greater optimization effectiveness.

5.2 τ-bench

Setup τ-bench (Yao et al., 2025) evaluates agents in multi-step tool-use environments. We focus on

the retail domain due to the availability of training data.

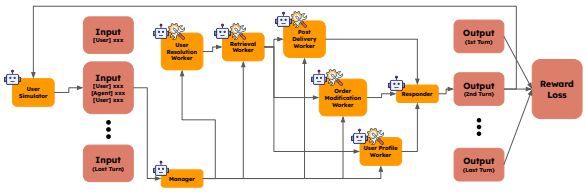

Figure 7: Multi-agent system tailored to τ-bench.

The system follows a manager–worker design (Figure 7). Workers handle user resolution, retrieval, order modification, post-delivery, and user profile tasks, while a responder generates outputs. A user simulator interacts with the system iteratively.

Verbal Loss We define a reward-based verbal loss at the conversation level, which includes the ground-truth tool-call trajectory, the agent-generated trajectory, and required response contents. The loss evaluates both tool-call correctness and whether system outputs contain all required information. Details are provided in Appendix B.2.

Metrics We evaluate performance using three conversation-level metrics. Action reward measures whether the sequence of tool calls matches the ground-truth trajectory. Output reward measures whether system responses contain all required information. Overall reward is defined as the product of the two, requiring both correct actions and complete responses.

Optimization Setup We use 10 training tasks and update prompts after each conversation. GPT-4o-mini is used as the user simulator due to budget constraints.

Results Results are shown in Table 2. Optimization consistently improves multi-agent performance over its pre-optimization baseline. Among the optimized variants, max of L1 norm yields the strongest overall performance for Qwen-3-32B, improving overall reward from 13.0 to 24.3, surpassing the strong single-agent baseline of 22.6. Mean of product with input also reaches an overall reward of 24.3 for Qwen-3-32B, mainly driven by a large improvement in action reward from 13.9 to 27.0. For Llama-3.3-70B-It, all optimized multi-agent variants improve over the pre-optimization baseline, with max of product with input achieving the best overall reward, increasing from 6.1 to 9.6. These results demonstrate the overall effectiveness of GBC for optimizing multi-agent systems, as it consistently improves overall reward across both backbone models and enables the Qwen-3-32B multi-agent system to outperform its strong single-agent counterpart.

Model	Action Reward	Output Reward	Overall Reward
Single-Agent			
Llama-3.3-70B-It	09.6	62.6	07.0
Qwen-3-32B	**27.8**	71.3	22.6
Multi-Agent (Before Optimization)			
Llama-3.3-70B-It	09.6	70.4	06.1
Qwen-3-32B	13.9	**78.3**	13.0
Multi-Agent (Optimized with Mean of L1 Norm)			
Llama-3.3-70B-It	11.3	70.4	08.7
Qwen-3-32B	17.4	73.0	13.9
Multi-Agent (Optimized with Max of L1 Norm)			
Llama-3.3-70B-It	13.9	73.0	09.6
Qwen-3-32B	**28.7**	**79.1**	**24.3**
Multi-Agent (Optimized with Mean of Product with Input)			
Llama-3.3-70B-It	12.2	72.2	08.7
Qwen-3-32B	27.0	**78.3**	**24.3**
Multi-Agent (Optimized with Max of Product with Input)			
Llama-3.3-70B-It	13.9	71.3	09.6
Qwen-3-32B	20.0	77.4	17.4

Table 2: Single-agent system and multi-agent system performance on τ-bench. The table shows the action rewards, output rewards, and overall rewards for the retail domain. **Best results** are bold and underlined; **second-best** are bold.

Figure 8: The occurrences of different error types detected by the optimizer under different connection weight formulae on τ-bench with Qwen-3-32B.

Error Analysis We identify five error types: tool misuse, retrieval/identification failure, unclear manager instructions, premature escalation, and incorrect explanations.

As shown in Figure 8, retrieval and identification

failures dominate across connection-weight settings. This reflects a central difficulty of τ-bench: successful task completion depends not only on selecting the right tool, but also on resolving the correct user, order, and task state across multiple turns. When this grounding step fails, later tool calls and final responses are likely to become incorrect even if the overall workflow is reasonable. Tool misuse also contributes to failures, indicating that action selection remains important. Premature escalation or stalling shows that agents sometimes fail to persist through the required tool-use procedure, while incorrect success or explanation affects whether the final response communicates the correct outcome. Overall, these errors highlight the long-horizon nature of τ-bench, where reliable retrieval, entity tracking, and inter-agent information transfer are necessary for both high action reward and high output reward.

6 Conclusion

In this work, we introduce Gradient-Based Connections (GBC), a novel framework for fine-grained attribution and optimization in multi-agent systems. By modeling a multi-agent workflow as a computational graph and leveraging gradient-based signals at the token level, GBC enables precise identification of how intermediate agent outputs influence downstream decisions. This formulation addresses a fundamental limitation of prior approaches—namely, the lack of effective credit assignment across agents—and provides a principled mechanism for diagnosing and improving multi-agent coordination.

Building on GBC, we develop AgentChord, an efficient optimization framework that integrates attribution-guided feedback with iterative prompt refinement. Through a prefix-based gradient computation strategy, AgentChord makes gradient-based attribution feasible for large-scale language models. Empirical results on MultiWOZ and τ-bench demonstrate that GBC consistently improves multi-agent performance across a range of metrics, and in many cases enables multi-agent systems to match or surpass strong single-agent baselines. Analysis of the MultiWOZ results further reveals that higher attribution quality is associated with greater optimization effectiveness. Furthermore, our analysis provides meaningful insights into error patterns, revealing both system-level weaknesses and task-intrinsic challenges.

Overall, our work highlights the importance of token-level, cross-agent credit assignment as a key component for advancing multi-agent systems. We believe GBC offers a general and extensible approach for future research on interpretable and optimizable multi-agent architectures.

Limitations

Despite its effectiveness, our approach has several limitations that warrant further investigation.

First, computational cost remains a concern. Although the prefix-based optimization reduces memory overhead, gradient-based attribution still requires multiple forward and backward passes through LLMs, making the approach expensive compared to purely black-box methods. This constraint limits scalability to larger systems or longer interaction horizons.

Second, GBC relies on the quality and design of the verbal loss function. Since the loss is task-specific and expressed in natural language, its effectiveness depends on how well it captures fine-grained errors. Poorly designed loss signals may lead to noisy or misleading attribution, reducing optimization effectiveness.

Third, while GBC provides token-level attribution, it still operates under a first-order approximation of influence (e.g., gradient-based signals). This may not fully capture complex nonlinear interactions between agents, especially in long multi-turn or highly entangled workflows.

Fourth, our experiments focus on specific benchmarks (MultiWOZ and τ-bench) and particular system architectures (e.g., manager–worker setups). Although results are promising, the generalization of GBC to other domains—such as open-ended reasoning, code generation, or large-scale autonomous agent systems—remains to be validated.

Finally, our analysis reveals that some error types—such as cross-domain errors, information omission, and retrieval failures—persist even after optimization, suggesting that these may stem from intrinsic task difficulty or limitations of current LLMs, rather than attribution alone.

Future work may explore more efficient gradient approximations, improved verbal loss design, integration with reinforcement learning, and extensions to dynamic or adaptive multi-agent topologies.

References

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *The Fourteenth International Conference on Learning Representations*.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. MultiWOZ - a large-scale multi-domain Wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026, Brussels, Belgium. Association for Computational Linguistics.

Yubo Dong, Xukun Zhu, Zhengzhe Pan, Linchao Zhu, and Yi Yang. 2024. VillagerAgent: A graph-based multi-agent framework for coordinating complex task dependencies in Minecraft. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 16290–16314, Bangkok, Thailand. Association for Computational Linguistics.

Mihail Eric, Rahul Goel, Shachi Paul, Abhishek Sethi, Sanchit Agarwal, Shuyang Gao, Adarsh Kumar, Anuj Goyal, Peter Ku, and Dilek Hakkani-Tur. 2020. MultiWOZ 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 422–428, Marseille, France. European Language Resources Association.

Aman Gupta, Anirudh Ravichandran, Ziji Zhang, Swair Shah, Anurag Beniwal, and Narayanan Sadagopan. 2024. Dard: A multi-agent approach for task-oriented dialog systems. *Preprint*, arXiv:2411.00427.

Lewis Hammond, Alan Chan, Jesse Clifton, Jason Hoelscher-Obermaier, Akbir Khan, Euan McLean, Chandler Smith, Wolfram Barfuss, Jakob Foerster, Tomáš Gavenčiak, The Anh Han, Edward Hughes, Vojtěch Kovařík, Jan Kulveit, Joel Z. Leibo, Caspar Oesterheld, Christian Schroeder de Witt, Nisarg Shah, Michael Wellman, and 25 others. 2025. Multi-agent risks from advanced ai. Technical Report 1, Cooperative AI Foundation.

Ting Han, Ximing Liu, Ryuichi Takanabu, Yixin Lian, Chongxuan Huang, Dazhen Wan, Wei Peng, and Minlie Huang. 2021. Multiwoz 2.3: A multi-domain task-oriented dialogue dataset enhanced with annotation corrections and co-reference annotation. In *Natural Language Processing and Chinese Computing: 10th CCF International Conference, NLPCC 2021, Qingdao, China, October 13–17, 2021, Proceedings, Part II*, page 206–218, Berlin, Heidelberg. Springer-Verlag.

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*.

Jun Liu, Zhenglun Kong, Changdi Yang, Fan Yang, Tianqi Li, Peiyan Dong, Joannah Nanjekye, Hao Tang, Geng Yuan, Wei Niu, Wenbin Zhang, Pu Zhao, Xue Lin, Dong Huang, and Yanzhi Wang. 2025. Rcr-router: Efficient role-aware context routing for multi-agent llm systems with structured memory. *Preprint*, arXiv:2508.04903.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. 2025. Agent lightning: Train any ai agents with reinforcement learning. *Preprint*, arXiv:2508.03680.

Melissa Z Pan, Mert Cemri, Lakshya A Agrawal, Shuyi Yang, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Kannan Ramchandran, Dan Klein, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. 2025. Why do multiagent systems fail? In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*.

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.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186, Bangkok, Thailand. Association for Computational Linguistics.

Chen Qian, Zihao Xie, YiFei Wang, Wei Liu, Kunlun Zhu, Hanchen Xia, Yufan Dang, Zhuoyun Du, Weize Chen, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2025. Scaling large language model-based multi-agent collaboration. In *The Thirteenth International Conference on Learning Representations*.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3145–3153. JMLR.org.

Xavier Sécheresse, Jacques-Yves Guilbert-Ly, and Antoine Villedieu de Torcy. 2025. Gaapo: Genetic al-

gorithmic applied to prompt optimization. *Preprint*, arXiv:2504.07157.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *Preprint*, arXiv:2308.08155.

Jinyu Xiang, Jiayi Zhang, Zhaoyang Yu, Xinbing Liang, Fengwei Teng, Jinhao Tu, Fashen Ren, Xiangru Tang, Sirui Hong, Chenglin Wu, and Yuyu Luo. 2025. Self-supervised prompt optimization. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 9017–9041, Suzhou, China. Association for Computational Linguistics.

Guowei Xu, Mert Yuksekgonul, Carlos Guestrin, and James Zou. 2025. metatextgrad: Automatically optimizing language model optimizers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. *Preprint*, arXiv:2309.03409.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. 2025. τ-bench: A benchmark for Tool-Agent-User interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*.

Eric Ye and Natasha Jaques. 2024. An efficient open world benchmark for multi-agent reinforcement learning. In *NeurIPS 2024 Workshop on Open-World Agents*.

Fanghua Ye, Jarana Manotumruksa, and Emine Yilmaz. 2022. MultiWOZ 2.4: A multi-domain task-oriented dialogue dataset with essential annotation corrections to improve state tracking evaluation. In *Proceedings of the 23rd Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 351–360, Edinburgh, UK. Association for Computational Linguistics.

Haofei Yu, Zirui Cheng, Zhaochen Hong, Kunlun Zhu, Jinwei Yao, Tao Feng, and Jiaxuan You. 2025. Research town: Simulator of research community.

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

Xiaoxue Zang, Abhinav Rastogi, Srinivas Sunkara, Raghav Gupta, Jianguo Zhang, and Jindong Chen. 2020. MultiWOZ 2.2 : A dialogue dataset with additional annotation corrections and state tracking baselines. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pages 109–117, Online. Association for Computational Linguistics.

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

Yaolun Zhang, Xiaogeng Liu, and Chaowei Xiao. 2025b. Metaagent: Automatically constructing multi-agent systems based on finite state machines. In *Forty-second International Conference on Machine Learning*.

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

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Gptswarm: language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

A Experiment Setting Detail

Unless otherwise specified, all experiments are conducted on a single compute node equipped with four NVIDIA A40 GPUs, 208 GB of system memory, and 16 CPU cores. We use the same hardware configuration for both the optimization and inference phases to ensure consistency across experimental runs. For local model serving, we enable FP8 quantization whenever supported by the target model and serving backend. We use GPT-4 as the optimizer model for prompt refinement throughout the optimization process. When using Qwen-3-32B, we turn off the thinking process in order to accelerate the experiments.

B Verbal Loss Prompts

This section presents the verbal-loss prompt templates used for different task scenarios, including MultiWOZ and τ-bench.

B.1 Verbal Loss Prompts for MultiWOZ

For MultiWOZ, we design two types of verbal loss: Joint Goal Accuracy (JGA) loss and Inform & Success loss. JGA loss is computed at the turn level. We present the prompt templates for both losses below.

B.1.1 JGA Loss

The prompt template for JGA loss is shown below:

Prompt Template for JGA Loss

The system has made the following user intention predictions:

`{prediction}`

The ground truth user intentions are:

`{ground_truth}`

The false positive predictions are:

`{false_positive}`

The false negative predictions are:

`{false_negative}`

When using this template, "prediction" is replaced with the JSON string of the predicted dialogue state for a given turn. Similarly, "ground_truth", "false_positive", and "false_negative" are replaced with the JSON strings of the ground-truth dialogue state, false-positive slot-value pairs, and false-negative slot names, respectively. An example dialogue state is shown below:

```
{
    "train-departure": "cambridge",
    "train-leaveAt": "11:00",
    "train-day": "wednesday",
    "train-destination": "stansted
        airport"
}
```

B.1.2 Inform & Success Loss

The prompt template for Inform & Success loss is shown below:

Prompt Template for Inform & Success Loss

The system has made the following queries:

`{provided_queries}`

The ground truth queries are:

`{requested_queries}`

The system has provided the following information:

`{provided_information}`

The ground truth information is:

`{requested_information}`

When using this template, "provided_queries" and "requested_queries" are replaced with the JSON strings of the queries generated by the system and the corresponding ground-truth queries. An example is shown below:

```
{
    "hotel": [
        {
            "area": "east",
            "internet": "yes",
            "parking": "no"
        },
        {
            "area": "east",
            "internet": "yes",
            "parking": "no"
        },
        {
            "area": "east",
            "internet": "yes",
            "parking": "no"
        }
    ]
}
```

Similarly, "provided_information" and "requested_information" are replaced with the information provided by the system and the corresponding ground-truth information. An example is shown below:

```
{
    "attraction": ["POST"],
    "taxi": ["PHONE"]
}
```

B.2 Verbal Loss Prompt for τ-bench

The prompt template for reward loss is shown below:

Prompt Template for Reward Loss

The ground truth actions are:

`{groundtruth_actions}`

The predicted actions are:

`{predicted_actions}`

Whether the actions match:

`{action_match}`

The required text strings in the responses are:

`{required_output_strings}`

The predicted responses are:

`{predicted_responses}`

Whether the responses match:

`{output_match}`

When using this template, "groundtruth_actions" and "predicted_actions" are replaced with the

ground-truth tool-call trajectories and the generated trajectories, respectively. An example is shown below:

```
[
    {
        "tool_name": "
            exchange_delivered_order
            _items",
        "tool_arguments": {
            "order_id": "#W9077205",
            "item_ids": ["3877338112"],
            "new_item_ids":
                ["2444431651"],
            "payment_method_id": "
                gift_card_7108145"
        }
    }
]
```

Similarly, "action_match" and "output_match" are replaced with the corresponding binary rewards, while "required_output_strings" and "predicted_responses" are replaced with the list of required text strings and the list of system utterances, respectively.

C Language Model as Optimizer

C.1 Pseudocode of Language Model as Optimizer

Algorithm 2 presents the pseudocode for using a language model as an optimizer.

Algorithm 2 Language Model as Optimizer

Require: Initial agent prompts P^0
Require: agent tools $\mathcal{A}$
Require: Prompt template $\mathcal{F}(\cdot)$
Initialize agent prompts $P \leftarrow P^{(0)}$
Initialize optimization history $H \leftarrow [\,]$
for $t = 1$ to T **do**
    Run the MAS on the training samples and collect latest prompts P^t, latest performance r^t, and latest attribution trajectories $\mathcal{T}^t$
    $P \leftarrow P^t$
    $r \leftarrow r^t$
    $\mathcal{T} \leftarrow \mathcal{T}^t$
    Append (P, r) to optimization history H
    $M \leftarrow \mathcal{F}(P, \mathcal{A}, H, \mathcal{T})$
    Query LLM with M to obtain prompt updates ΔP
    Update prompts: $P \leftarrow P + \Delta P$
end for
return P

C.2 Prompt of Language Model as Optimizer

At each optimization step, the optimizer receives an instruction prompt and an input prompt. We present the corresponding prompt templates below. The instruction prompt is as follows:

Prompt for Optimizer Instruction

You are an expert in analyzing multi-agent systems and providing insights on how to improve agent performance through better prompts.
You will be given the structure of a multi-agent system, including the names of different agents and their current prompts.
You will be provided with some inference trajectories from the multi-agent system. Each trajectory consists of a sequence of outputs from different agents. Each output is conditioned on the previous outputs.
At the end of each trajectory, there is a comparison between the final output and the expected output.
You will also receive an optimization history that contains information about the agents and their prompts.
Your task is to analyze these trajectories and provide insights on how the agents can avoid the mistakes and achieve better results by improving their prompts. Note that the order of the agents is fixed. So you should not suggest changing the order of the agents.
If there's no **Warning** section in the prompt, you can add such a section at the end of the prompt. You can add sentences to warn the agents about the mistakes in the trajectories. For example, if an agent often misses certain tool calls, you can encourage the agent to use certain tools under certain situations; if an agent often misses certain information in tool call inputs, you can warn the agent to pay attention to that information. You should format the warnings in bullet points in markdown format. For each warning, you should attach a failure case from the trajectories as an example.
Note that the toolkit of each agent might be different. So you should not suggest using tools that are not available to the agent.
However, the content within the {DONT_CHANGE_HEADER} and {DONT_CHANGE_FOOTER} tags and the tags themselves should be kept unchanged and preserved in the new prompt.
You should also note that the causality in some trajectories is noisy. So you should not

take the noisy causality into account.

Procedure
You should follow these steps: 1. Identify the agents that need to improve their prompts. 2. For each selected agent, suggest a new prompt that could lead to better results.
Input Format
You will receive the optimization history and trajectories in the following format: The structure of the multi-agent system:

```
{
  "agent_name_1": "Current prompt
      for agent 1",
  "agent_name_2": "Current prompt
      for agent 2",
  ...
}
```

The tools available to each agent:

```
{
  "agent_name_1": [... tool
      descriptions ...],
  "agent_name_2": [... tool
      descriptions ...],
  ...
}
```

The optimization history and the corresponding performance:

```
[
  {
    "prompts": {
      "agent_name_1": "Prompt for
          agent 1",
      "agent_name_2": "Prompt for
          agent 2",
      ...
    },
    "performance": "A brief
        description of the
        performance of the multi-
        agent system, including the
        expected output and the
        actual output.",
  },
  {
    "prompts": {
      "agent_name_1": "Prompt for
          agent 1",
      "agent_name_2": "Prompt for
          agent 2",
      ...
    },
    "performance": "A brief
        description of the
        performance of the multi-
        agent system, including the
        expected output and the
        actual output.",
  },
  ...
]
```

The inference trajectories:

```
[
  "agent_1: output_1 -> agent_2:
      output_2 -> ... -> agent_n:
      final_output -> loss:
      comparison_of_final_output_and
       _expected_output",
  "agent_1: output_1 -> agent_2:
      output_2 -> ... -> agent_n:
      final_output -> loss:
      comparison_of_final_output_and
       _expected_output",
  ...
]
```

Output Format You should output a JSON object with the following structure:

```
{
  "reasoning": "Your reasoning about
       the agents and their prompts
      .",
  "agent_prompts": {
    "agent_name_x": "New prompt for
        agent x containing the **
        Warning** section.",
    ...
  }
}
```

The input prompt template is as follows:

Prompt Template for Inform & Success Loss

The structure of the multi-agent system:

`{agent_structure}`

The tools available to each agent:

`{agent_tools}`

The optimization history and the corresponding performance:

`{optimization_history}`

The inference trajectories:

`{inference_trajectories}`

D Time Cost

Table 3 reports the running time of completing 10 steps of optimization under different connection weight settings on MultiWOZ and τ-bench. Overall, Llama-3.3-70B-It requires substantially longer running time than Qwen-3-32B across both benchmarks, which is expected given the larger model size. On MultiWOZ, the running times are relatively stable across different connection weight

choices: Llama-3.3-70B-It takes around 16.3–16.7 hours, while Qwen-3-32B takes around 8.0–9.5 hours. On τ-bench, the runtime varies more noticeably for Llama-3.3-70B-It, ranging from 10.3 to 16.3 hours depending on the connection weight, whereas Qwen-3-32B remains consistently around 5–6 hours. These results indicate that the choice of connection weight does not introduce a substantial additional computational burden, and the overall runtime is primarily determined by the benchmark and the underlying model.

Model	Connection Weight	Time
MultiWOZ		
Llama-3.3-70B-It	Mean of L1 Norm	16h39m20s
	Max of L1 Norm	16h40m46s
	Mean of Product with Input	16h26m19s
	Max of Product with Input	16h19m55s
Qwen-3-32B	Mean of L1 Norm	08h45m25s
	Max of L1 Norm	08h00m16s
	Mean of Product with Input	09h27m18s
	Max of Product with Input	08h26m49s
τ-bench		
Llama-3.3-70B-It	Mean of L1 Norm	16h20m07s
	Max of L1 Norm	12h52m50s
	Mean of Product with Input	14h47m31s
	Max of Product with Input	10h16m37s
Qwen-3-32B	Mean of L1 Norm	05h09m28s
	Max of L1 Norm	05h59m47s
	Mean of Product with Input	05h06m50s
	Max of Product with Input	06h04m16s

Table 3: Running time of optimization under different connection weight settings on MultiWOZ and τ-bench.