FlowXpert: Expertizing Troubleshooting Workflow Orchestration with Knowledge Base and Multi-Agent Coevolution

Binpeng Shi
Nankai University
Tianjin, China

Yu Luo
Jingya Wang
Nankai University
Tianjin, China

Yongxin Zhao
Nankai University
Tianjin, China

Shenglin Zhang
Nankai University
Tianjin, China

Bowen Hao
Nankai University
Tianjin, China

Chenyu Zhao
Nankai University
Tianjin, China

Yongqian Sun
Nankai University
Tianjin, China

Zhi Zhang
Ronghua Sun
Haihua Li
Huawei Cloud
Dongguan, China

Wei Song
Xiaolong Chen
Jingbo Miao
Huawei Inc.
Nanjing, China

Dan Pei
Tsinghua University
Beijing, China

Abstract

Incident management remains a critical yet challenging task for large-scale cloud services. Most cloud service providers abstract troubleshooting into predefined workflows for different incidents, offering step-by-step guidance. However, manually crafting workflows is resource-consuming and knowledge-intensive, hindering large-scale deployment. Most automated techniques for workflow orchestration rely on large language models (LLMs) to handle complex tasks but overlook key aspects of troubleshooting, including complex expertise, domain requirements, and the reliability of AI feedback. These limitations undermine workflow quality. Therefore, we propose FlowXpert, a novel framework for troubleshooting workflow orchestration. Leveraging LLMs, it first builds a knowledge base centered on incident-aware nodes to precisely depict expertise. Then, fed into AI feedback and synthetic preference data, reinforcement learning is applied to refine the workflow generator and evaluator. To assess troubleshooting workflows, we introduce OpsFlowBench based on Huawei Cloud's datacenter switch operation documents. Benchmark tests under the tailored STEPScore metric validate its effectiveness. Furthermore, during a 10-week deployment in Huawei Cloud's datacenter network, FlowXpert provided valuable support to both on-call engineers and AI executors, as evidenced by empirical data and case study.

CCS Concepts

• **Software and its engineering → Software maintenance tools**.

Keywords

Troubleshooting, Workflow Orchestration, Incident Management, Large Language Model

1 Introduction

Huawei Cloud's datacenter network (DCN) hosts more than $O(10^6)$ servers and $O(10^5)$ switches across 17 regions and 63 availability zones worldwide. Each month, the system generates over 20,000 incident tickets, posing a significant threat to cloud service reliability [3–5]. At such a large scale, effective and efficient incident management becomes more and more essential. Nowadays, most cloud service providers embrace process automation [6, 22, 40, 42], abstracting troubleshooting into workflows for different incidents, which follow a structured sequence of core steps. The primary customer for workflows consists of on-call engineers (OCEs) and AI executors (Executors). (1) For OCEs, workflows offer step-by-step guidance, including operations, commands, and data queries, thereby reducing expertise demands and boosting efficiency. Additionally, as standardized carriers of expertise, workflows facilitate knowledge sharing. When encountering novel incidents, past workflows of similar cases could serve as valuable references. (2) Moreover, Huawei Cloud engineers have transformed common workflows into executable scripts, enabling automated incident management and reducing the excessive burden on OCEs. Moving forward, Executor, an AI agent equipped with tool invocation and result analysis capabilities, is expected to streamline this process by directly interpreting workflows, eliminating the need for manual script conversion [42].

Traditionally, workflows are manually crafted, demanding significant effort and deep expertise. This resource-consuming and knowledge-intensive process struggles to keep pace with the growing number and complexity of incident scenarios, hindering large-scale development and deployment of AI executors. Recent advancements in large language models (LLMs) demonstrate strong potential in understanding natural language and handling complex tasks across various domains [32, 44, 56], paving the way for automatic high-quality workflow generation.

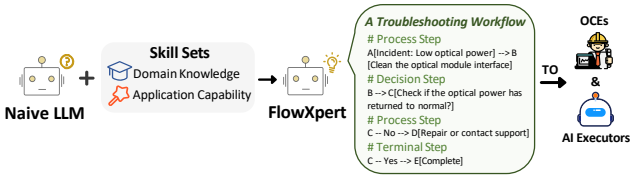

Figure 1: From naive LLM to workflow generator

As depicted in Fig. 1, for the transformation of naive LLMs into high-quality workflow generators, both domain knowledge and application capabilities are indispensable [18, 21, 27, 29, 53]. (1) **Support of domain knowledge**. It ensures the knowledge used in task-solving comes not from out-of-domain corpora but rather from expertise accumulated within the domain. To prevent hallucinations and noise, comprehensive and precise knowledge must be

provided, typically in three forms. The first is action space, where application programming interfaces (APIs) define the executable operations within a given domain [6, 16, 54]. These APIs, developed and tested by experts, exhibit high accuracy and reliability before being made available to LLMs. The other two forms involve vector-based [25] and graph-based [14] retrieval of relevant documents. Vector indexing enables efficient semantic matching across large-scale texts, ensuring broad coverage of relevant information. Graph indexing captures complex relationships, particularly upstream and downstream dependencies between documents, thereby enhancing retrieval depth. (2) **Alignment of application capability**. It guides LLMs to faithfully follow and fully explore domain knowledge, unlocking their workflow orchestration capability. Supervised fine-tuning (SFT) is a common approach [16]. It primarily relies on language modeling fine-tuning like next-token or mask prediction yet often lacks targeted optimization for workflow orchestration. In contrast, reinforcement learning (RL) incorporates a feedback mechanism to directly enhance task performance, allowing the model to iteratively refine and improve its knowledge application [26, 45]. Moreover, some techniques [24, 45] utilize AI as a feedback source in RL, further reducing the need for human intervention.

When equipped with domain knowledge support and aligned knowledge application capability, LLMs specialize in workflow orchestration. However, in troubleshooting, three domain-specific challenges still exist:

C1: Complexity of troubleshooting expertise. Generally, OCEs document operation experiences like incident indicators, root causes, mitigation steps, *etc.* However, defining APIs based on these records to provide domain knowledge is suboptimal, as the process is labor-intensive. Furthermore, the constrained scope of API-defined spaces cannot fully capture the complexity and variability of troubleshooting expertise, which restricts LLMs from orchestrating only simple, small-scale workflows [16, 52]. Vector indexing also struggles to integrate distant textual information, limiting the depth of knowledge extraction [14]. Meanwhile, graph indexing faces granularity issues of entities and relationships, which often overlook the granularity required for troubleshooting [27, 51]. These limitations make it challenging to precisely depict complex troubleshooting expertise.

C2: Compliance of workflow orchestration with domain requirements. In troubleshooting, an effective workflow should comprehensively recall all key steps. Some redundancy or minor imperfections are acceptable and easily adjustable. Additionally, workflows must meet specific requirements such as readability and executability. That is, they should avoid confusion for OCEs and Executors while successfully guiding troubleshooting procedures.

C3: Reliability of AI feedback. RL with AI feedback is a common approach to improving workflow generation quality [26, 45]. In production environments, given concerns about resource cost and data privacy, customized open-source models are typically adopted as AI evaluators. However, static open-source LLMs have limited capability for providing feedback, especially in knowledge-intensive troubleshooting, where they may even produce incorrect feedback. None of the existing techniques [24, 26, 45] have accounted for this when leveraging AI evaluators.

In this work, we propose FlowXpert, a framework for automated orchestration of troubleshooting workflows. Specifically, FlowXpert consists of two modules: (1) *Knowledge Base Construction*. We convert operation documents into vector and graph databases to offer comprehensive and deep domain knowledge support (C1). The graph base is structured around a core of incident-aware nodes. Assisted by LLMs, the construction process includes incident extraction, node filling, merging, and refinement. (2) *Multi-Agent Coevolution*. This module instantiates two LLM-driven agents, the Planner and the Scorer, responsible for orchestrating and evaluating troubleshooting workflows respectively. To align workflows with domain requirements (C2), we fine-tune the Planner using Proximal Policy Optimization (PPO) [39], guided by multi-dimensional scores from the Scorer. To improve AI feedback reliability (C3), we synthesize preference data controlled by contextual richness, then fine-tune the Scorer using Direct Preference Optimization (DPO) [35]. The Planner and Scorer collaborate and coevolve to ensure precise application of domain knowledge, thereby producing high-quality workflows.

Our contributions are summarized as follows:

- We propose FlowXpert, a novel framework that orchestrates troubleshooting workflows by integrating domain knowledge support and aligned knowledge application.
- For high-quality workflow generation, we (1) define a domain ontology to guide the knowledge base construction, which converts troubleshooting expertise into precise incident-aware nodes (Sec. 4.1), (2) implement multi-agent coevolution through PPO and DPO tuning (Sec. 4.3), (3) design a preference data synthesis method controlled by contextual richness, improving the AI evaluator's discernment capability (Sec. 4.3).
- To evaluate models' capability to orchestrate troubleshooting workflows, we introduce STEPScore, a metric designed around core characteristics of workflows, and conduct extensive benchmark tests based on real-world incidents from Huawei Cloud datacenter switches (Sec. 5.1). The results demonstrate FlowXpert's effectiveness (Sec. 5.2).
- During a 10-week deployment in the Huawei Cloud's DCN, our framework contributed a lot to both OCEs (Sec. 6.1) and Executors (Sec. 6.2), recorded through empirical data and case study.

2 Related Work

Support of domain knowledge. When adopting LLMs for domain-specific tasks, comprehensive and in-depth knowledge support is essential, typically encompassing three forms. (1) Action space. ReAct [52] reveals the integration of reasoning and acting to generate task-solving trajectories. Building on this, ToolLLaMA [34] and T-eval [10] demonstrate LLMs' capability to employ tools for domain-specific tasks. In production settings, the tools are virtualized as a set of APIs, which are meticulously developed and rigorously tested to equip LLMs with extensive domain knowledge. Several techniques [6, 16, 54] enable LLMs to learn, select, and invoke APIs to generate workflows and solve tasks. However, defining an API-based action space to sketch expertise is labor-intensive and inherently limited in scope. (2) Vector indexing. To provide comprehensive knowledge, vector indexing enables retrieval based on semantic similarity. Nevertheless, its linear structure hampers the establishment of long-range associations between texts [23, 25, 41]. (3) Graph indexing. In response, Graph RAG [14] constructs knowledge graphs by extracting entities and relationships, identifying communities, and

establishing knowledge links with a global perspective. Further techniques enhance knowledge depth by extending graphs through multi-hop connections [9], brain-inspired modeling [19], and topic-aware retrieval [30, 51]. However, manual graph construction lacks scalability, while automated methods often miss the optimal granularity for troubleshooting, leading to either excessive noise or information loss. To harmonize the strengths of both approaches, emerging techniques combine vector and graph bases to provide domain knowledge [27, 38]. Yet, granularity issues in graph bases still hinder the precise sketch of troubleshooting expertise. This is the work of FlowXpert's Module 1.

Alignment of knowledge application. Aligning LLMs with domain knowledge applications is crucial for tailoring them for specialized tasks. StateFlow [49] integrates finite state machines to explicitly define workflows, enabling better control over complex problem-solving. Certain techniques [26, 54] leverage LLMs' in-context learning by incorporating human feedback into prompts to refine generated workflows. Although effective and straightforward, these techniques rely on human intervention, limiting adaptability to diverse scenarios and restricting LLM adjustments. In contrast, fine-tuning presents a more robust strategy for teaching LLMs knowledge application patterns. SFT is commonly utilized to align LLMs with desired outputs. For example, WorkflowLLM [16] fine-tunes Llama with supervised learning on a large set of synthetic standard workflows. However, SFT focuses on language modeling rather than task-specific objectives. As a result, SFT tends to memorize training data, whereas RL generalizes in out-of-distribution data by directly optimizing for task performance [11]. AutoFlow [26] and PEER [45] utilize RL to refine workflow generators. Specifically, AutoFlow employs PPO based on workflow execution feedback, while PEER synthesizes preference data using AI feedback and then applies DPO fine-tuning. For the latter solution, extensive works [17, 24, 43, 48] have shown that AI feedback can match or even surpass human annotations while reducing manual workload. However, in automated troubleshooting, little attention has been given to how RL enhances knowledge application or whether AI feedback aligns with domain-specific requirements. This challenge is precisely what FlowXpert's Module 2 aims to address.

3 Motivation

This section explains our motivation in two aspects: *usage* and *acquisition*. The former emphasizes the practical utility and unrealized potential of workflows in production, while the latter outlines the process, costs, and principles for acquiring workflows.

3.1 Workflow Usage

As illustrated on the right side of Fig. 1, workflows comprise a sequence of steps for incident resolution. These steps are recorded in natural language, encompassing instructions, commands, queries, code snippets, *etc.* Logically, they can be classified into three types [26]: (1) Process step. It defines the action to be executed. (2) Decision step. It introduces a branch based on specific conditions. (3) Terminal step. It marks the completion. We commonly use Mermaid [1], a Markdown-like syntax, to represent structured workflows.

Workflows play a critical role in production. Since 2021, Huawei Cloud's OCEs have developed workflows for 189 distinct incident types across domains such as network, hardware, interface, and system. Among these, 79 high-frequency incidents have been automated into scripts. When these incidents occur, OCEs require no manual intervention; the program automatically generates execution results and analysis conclusions step by step. For incidents not equipped with scripts, OCEs receive workflow recommendations of similar cases. This provides a reference for every operation, reducing reliance on expertise and boosting efficiency. Notably, with the aid of workflows and scripts, the average incident resolution time has dropped from 24 minutes to 9 minutes, a 62.5% reduction. Despite more incidents from expanded business, labor costs remain unchanged.

3.2 Workflow Acquisition

The current workflow acquisition process primarily depends on manual effort. At Huawei Cloud, OCEs routinely review operation documents from recent incident handling to derive workflows. Developing a workflow for a single incident requires a team of seven OCEs, including two experts, and takes approximately seven hours. The process is as follows: (1) *Summarization* (two hours). Collecting typical cases, analyzing root causes, and identifying key metrics. (2) *Formulation* (two hours). Defining standard incident-handling steps and assessing coverage and effectiveness. (3) *Orchestration* (three hours). Designing and testing workflows. On one hand, this is a resource-consuming and knowledge-intensive task. As the business expands, manually updating and maintaining workflows struggle to keep pace with the growing complexity and volume of incident scenarios. On the other hand, a comprehensive survey on OCEs reveals that factors like inconsistent standards, biased expertise, unclear expressions, and redundant or omitted steps challenge the quality of manually created workflows. These issues may mislead OCEs or Executors, potentially causing errors and greater losses in incident handling. In a word, OCEs call for an automated approach to orchestrate workflows from operation documents.

Usually, a high-quality workflow adheres to the following principles: fidelity to domain knowledge in operation documents, coverage of key steps, and readability and executability for both OCEs and Executors, *etc.* Despite these qualitative guidelines, there are few dedicated metrics for quantifying workflow quality. Task success rate is utilized as an indirect metric [6, 16, 26]. However, many operations like physically cleaning fan dust, involve long execution chains, making timely feedback impractical. Additionally, text generation metrics like BLEU [33] and ROUGE [28] fail to capture the core characteristics of workflows, *i.e.*, task-solving through multiple key steps. The absence of a dedicated evaluation system hinders the advancement of automated techniques for workflow generation.

> **INSIGHT:** Workflows play a critical role in troubleshooting, which urgently needs to shift from manual creation to automated orchestration. Additionally, a dedicated evaluation metric would be beneficial.

4 Methodology

As detailed in Fig. 2, the offline process of FlowXpert is dedicated to transforming naive LLMs into high-quality workflow generators.

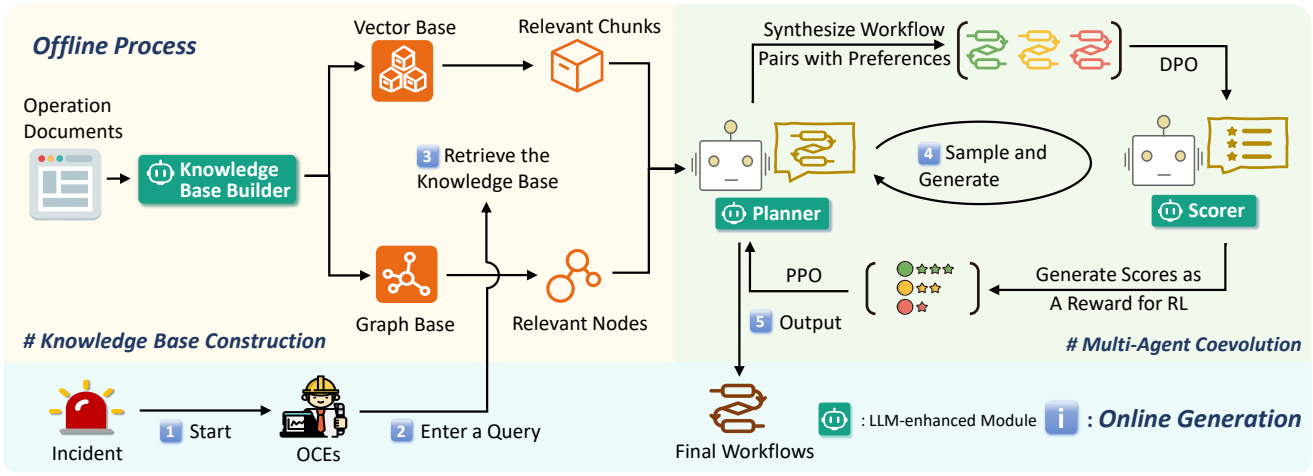

Figure 2: The framework of FlowXpert

Specifically, *Module 1, Knowledge Base Construction*, builds vector and graph bases by parsing operation documents that cover typical cases, incident indicators, root causes, and mitigation steps, thereby providing troubleshooting expertise for workflow orchestration. *Module 2, Multi-Agent Coevolution*, focuses on fine-tuning two LLM-driven agents: Planner and Scorer, responsible for generating workflows and assessing their quality, respectively. First, PPO tuning, guided by multi-dimensional AI feedback from Scorer, enhances the Planner. Second, we synthesize workflow pairs with preferences controlled by the richness of domain knowledge. DPO is then employed to refine the Scorer's judgment capability. These fine-tuning strategies collectively enhance the application of domain knowledge from Module 1. For online generation, when handling an incident ticket, FlowXpert first retrieves contextual knowledge from knowledge bases according to OCEs' queries. Then the Planner and Scorer engage in sampling and generation, automatically producing workflows for incident management.

4.1 Knowledge Base Construction

This module systematically integrates contextual knowledge from both vector and graph bases.

Construction. The vector indexing adheres to the standard paradigm [25], where raw documents are segmented into chunks due to the LLMs' context size limitations. An encoder converts these chunks into embeddings, which are then stored in a vector database. The graph indexing is designed for this work. Specifically, it follows four steps with the assistance of an LLM-enhanced knowledge base builder: predefining incident-aware nodes, extracting incidents from chunks, filling in nodes, and merging and refining them across chunks, as shown in Fig. 3 and detailed below. All prompt templates can be found in Fig. 7 of Appendix B.

Step 1: Predefining Incident-Aware Nodes. The key to constructing a graph knowledge base (KG) lies in triplet extraction. However, irrelevant entities or relations with poor granularity introduce noise or omit critical information. Instead, ontology offers a standardized framework for knowledge representation through detailed concept

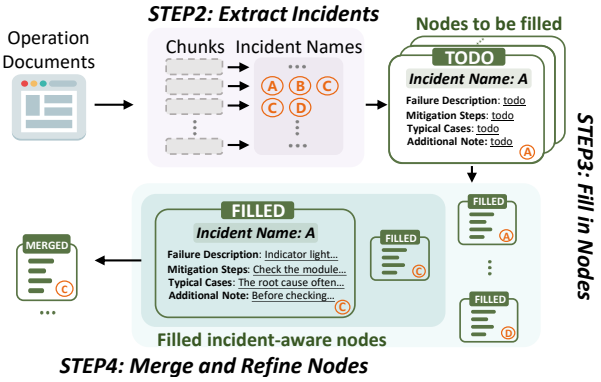

Figure 3: The process of graph base construction

definitions and relation modeling at a higher level, thereby mitigating granularity issues in knowledge management while facilitating knowledge sharing, updating, and expansion [7, 12, 15, 31, 47].

Therefore, FlowXpert constrains triplet extraction within a predefined troubleshooting ontology, ensuring a knowledge graph that remains highly focused on the operation domain. Specifically, we abstract the key elements of incident management into five concepts: $Concepts = \{Incident, Failure\ Descprition, Mitigation\ Steps, Typical\ Cases, Additional\ Note\}$. Among them, *Incident* serves as the central concept, while the others characterize its attributes from various perspectives. Accordingly, we define the ontology relations: $Relations = \{(Incident, has\ attributes, Attrs)\}$, where $Attrs = Concepts \setminus \{Incident\}$. Leveraging the concepts and relationships, we restructure the triples in KG into multiple Incident-Aware Nodes: $node_i = \{(incident_i, has\ attributes, attrs_{ij})\}$. The $incident_i$ and $attrs_{ij}$ denote the instances of *Incident* and *Attrs*, respectively. Thus, KG is conceptualized as a set of incident-aware nodes, each node resembling a blank form to be filled, with the incident name as the primary key and the other fields as incident's attributes tailored to OCEs' interests and concerns. Consequently, triplet extraction in KG construction is elevated to the level of

incident-aware node completion, providing proper granularity for knowledge exploration in operation documents.

Step 2: Extracting Incidents from Chunks. Completing all contents of a node at once remains challenging. So we start with the incident name acquisition. Given the context length limitation of LLMs, we split the raw document into K chunks and instruct LLMs to extract all incident instances $a_i^{(k)}$, *i.e.*, incident names, for each chunk k.

Step 3: Filling in Incident-Aware Nodes. With determined incident names, this step identifies the remaining attributes of the nodes like filling in forms. Specifically, LLMs extract other attributes from the chunk in a few-shot setting, *i.e.*, $attrs_{ij}^{(k)}$ associated with $a_i^{(k)}$. Notably, chunking may disperse the information of the same incident across multiple chunks. To mitigate the risk of missing or mismatching attributes due to absent incident instances, we fill the nodes with the incidents extracted from the current and the most recent previous chunk. The final node set is $S^{(k)}=\{node_1^{(k)},\ldots,node_n^{(k)}\}$, where n denotes the node number extracted from chunk k.

Step 4: Merging and Refining Nodes across Chunks. In *Step 3*, incorporating incidents from the previous chunk mitigates information loss but leads to notable node redundancy. To address this, we merge and refine nodes of the same incident type. Utilizing the strong semantic capability of LLMs, this step merges, restructures, and refines node content. The final output is a knowledge graph consisting of incident nodes, *i.e.*, $KG = \{node_i \mid i = 1, ..., N\}$, where N denotes the number of all incident types. Each node corresponds to an incident type, consolidating all relevant information from the source documents into four attributes: failure descriptions, mitigation steps, typical cases, and additional notes.

Retrieval. When an incident is triggered online, a query $Q^{(T)}$ is generated, typically including the incident name and description. The semantic similarity is then calculated between $Q^{(T)}$ and the vector indices of each chunk, as well as the encoded incident name of each node. Relevant domain knowledge is ranked by similarity scores, *i.e.*, $C^{(T)}=\{chunks_{topK}^{(T)}, nodes_{topN}^{(T)}\}$, where $chunks_{topK}^{(T)}$ and $nodes_{topN}^{(T)}$ represent the $topK$ and $topN$ most relevant chunks and nodes in natural language, respectively.

4.2 Agent Roles and Their Collaboration

We employ LLM-driven agents to simulate the workflow generator and evaluator. Each agent specializes in a single task, collaborating to drive high-quality workflow orchestration. All prompt templates can be found in Fig. 8 of Appendix B.

Planner and Scorer. (1) The *Planner* agent orchestrates workflows using retrieved knowledge from the previous module, $W^{(T)} = Planner(Q^{(T)}, C^{(T)})$. (2) The *Scorer* agent evaluates how well the generated workflows align with domain requirements. It assigns multidimensional scores, covering Relevance (consistency with contextual knowledge), Coverage (completeness of necessary steps), Accuracy (correctness of steps), Coherence (logical flow), and Conciseness (clarity, brevity, and ease of execution). The overall workflow quality is quantified by averaging these five scores, $S^{(T)} = Scorer(Q^{(T)}, C^{(T)}, W^{(T)})$. The score $S^{(T)}$ is utilized for both offline fine-tuning and online generation.

Best-of-N Sampling. During online generation, the Planner orchestrates N workflows $W^{(T)}$ for a given query $Q^{(T)}$, selecting the best one based on the scores $S^{(T)}$ from the Scorer. This setup simply and directly scales inference, enabling broad exploration by the Planner while ensuring quality by the Scorer.

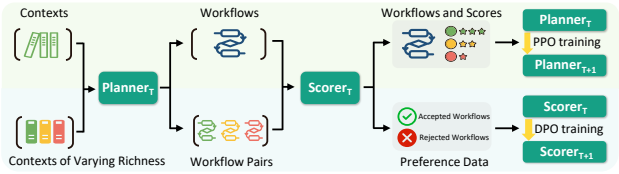

Figure 4: One iteration of multi-agent coevolution

4.3 Multi-Agent Coevolution

Planner and Scorer are not born experts but evolve through iterative fine-tuning. This refinement becomes especially critical when resource constraints and data privacy concerns drive the need for open-source, lightweight LLMs. Next, we outline an iteration of the coevolution process, as depicted in Fig. 4.

PPO for Planner. Inspired by [17, 24, 43, 48], AI feedback can achieve comparable performance to human feedback in reinforcement learning, significantly reducing human effort. FlowXpert employs PPO tuning to enhance the Planner's workflow orchestration, guided by multidimensional domain-specific feedback from the Scorer. The main objective is as follows:

$$\begin{aligned} L^{CLIP} &= \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right] \\ \hat{A}_t &= \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \\ \delta_t &= r_t + \gamma V(s_{t+1}) - V(s_t) \end{aligned} \tag{1}$$

where the term $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ represents the policy ratio, indicating the likelihood ratio between the current and previous iteration Planner in generating each $token_t$ during the production of workflow $W^{(T)}$ based on the prompt; $\hat{A}_t$ is the advantage estimation, derived from the immediate reward r_t and the state-value function $V(s_t)$; The sentence-level score $S^{(T)}$ from the Scorer is assigned as the immediate reward for the last token, while the KL divergence penalty between new and old policies adjusts the immediate rewards for other tokens; The clip function and ϵ constrain policy update magnitudes to ensure training stability.

DPO for Scorer. Given the importance of AI feedback reliability, FlowXpert strengthens the Scorer's workflow evaluation by combining data synthesis and DPO tuning.

Step 1: Synthesizing Workflow Pairs with Preferences Controlled by Knowledge Richness. In practice, identifying and correcting a small number of redundant steps is relatively straightforward, while OCEs prioritize comprehensive coverage of key steps. We generally consider the workflow quality to be positively correlated with key steps, determined by the richness of contextual knowledge. Therefore, for a query $Q^{(T)}$, we provide the Planner with three levels of contexts: complete and correctly ordered recommendations, complete but reversed ordered recommendations, and no context. The three levels correspond to the generation of workflows with varying quality. By pairing them, we obtain workflow pairs with preferences: $\mathcal{P} = \{(W_g^{(T)}, W_f^{(T)}), (W_g^{(T)}, W_p^{(T)}), (W_f^{(T)}, W_p^{(T)})\}$.

The subscripts g, f, and p denote quality levels: good, fair, and poor, respectively. Additionally, we employ the Scorer to validate their quality, discarding any pairs misaligned with high or low scores.

Step 2: DPO Tuning. Based on the synthesized preference data $\mathcal{D} = \{(X, W_{acc}, W_{rej}) \mid (W_{acc}, W_{rej}) \in \mathcal{P}\}$, we perform DPO tuning. X is the prompt template integrating the query and normal contextual knowledge. W_{acc} and W_{rej} denote the higher-quality and lower-quality workflows, respectively, within each pair of $\mathcal{P}$. The loss function is as follows:

$$L_{DPO}(\pi_\theta; \pi_{ref}) = -\mathbb{E}_{(X_T, W_{acc}, W_{rej})\sim D} \left[\log \sigma\Big(\beta \log \frac{\pi_\theta(W_{acc} \mid X)}{\pi_{\text{ref}}(W_{acc} \mid X)} - \beta \log \frac{\pi_\theta(W_{rej} \mid X)}{\pi_{\text{ref}}(W_{rej} \mid X)}\Big)\right] \quad (2)$$

where π_θ and π_{ref} denote the Scorer of the current and previous iteration, respectively; the scaling factor β measures errors in ranking results and accounts for the KL constraint.

Planner and Scorer undergo iterative tuning via PPO and DPO, initialized from seed LLMs. The process relies solely on queries from the training set, without the need for standard workflows. Through multiple rounds of coevolution, Planner's generation quality and Scorer's feedback effectiveness progressively improve together.

5 Experiment

In this section, we address the following research questions:
RQ1: How to evaluate the workflow orchestration capability in the operation domain?
RQ2: How well does FlowXpert perform?
RQ3: Does each component contribute to FlowXpert?

5.1 RQ1: Evaluation System

Dataset: OpsFlowBench. To address the absence of task-specific benchmarks for troubleshooting workflow planning, we construct OpsFlowBench, a dataset derived from Huawei Cloud datacenter switch operation documents. These documents encapsulate domain expertise, detailing real-world incident descriptions, indicators, root causes, and mitigation procedures. The dataset comprises 252 user queries from 4 scenarios (hardware, interface, network, top) spanning 56 major incident types. Each query is paired with a standard troubleshooting workflow, $case_i = (Q_i, W_i)$. The distribution of cases across four scenarios is 83, 56, 31, and 82, respectively. The above construction follows a multi-stage process: initial workflow generation via GPT-4o, followed by manual refinement and validation conducted by a team of three graduate researchers specializing in Artificial Intelligence for IT Operations (AIOps) and experienced OCEs at Huawei Cloud.

Metric: STEPScore. Common text generation metrics such as BLEU [33] and ROUGE [28] are often inadequate for assessing workflow quality, as they fail to capture the structured nature of workflows, which consist of a sequence of core steps. Inspired by BERTScore [55], we propose STEPScore as a specialized evaluation metric. Specifically, both the generated and reference workflows are parsed into key step sets, S_g and S_r, respectively. (1) For each step s_i in S_g, we compute its maximum cosine similarity with all steps in S_r, denoted as p_i. The average of p_i defines the workflow's precision, indicating how closely the generated steps match the standard steps. $Precision = \frac{1}{|S_g|}\sum_{s_i \in S_g} \max_{s_j \in S_r} \cos(E(s_i), E(s_j))$. (2) For each step s_j in S_r, we compute its maximum cosine similarity with all steps in S_g, denoted as p_j. The average of p_j defines the workflow's recall, indicating how well the standard steps are retrieved in the generated steps. $Recall = \frac{1}{|S_r|}\sum_{s_j \in S_r} \max_{s_i \in S_g} \cos(E(s_i), E(s_j))$. The $F1$ score is the harmonic mean of $Precision$ and $Recall$. Notably, execution-related metrics like pass rate are excluded in benchmark tests, as certain operations (*e.g.*, physically cleaning fan dust) cannot be timely assessed in offline settings. Instead, such metrics as acceptance rate are applied in online deployment (Sec. 6.1).

Implementation details, including software environment, hardware configurations, training procedures, dataset split, and hyperparameters, are provided in Appendix A.

5.2 RQ2: Overall Performance

We evaluate FlowXpert through three comparisons:

- **Expertise Sources**. We examine three knowledge retrieval methods: zero-shot, VectorRAG [25], and GraphRAG [14]. The generation module is the same as FlowXpert but without fine-tuning.
- **Tuning Approaches**. We compare different fine-tuning strategies, including supervised fine-tuning (SFT) and reinforcement learning with GPT-4o feedback (RL_GPT4o). Additionally, CoT [46] serves as a baseline without fine-tuning. All methods leverage FlowXpert's full knowledge base.
- **Seed LLMs**. Given the trade-off between cost and Chinese comprehension, we experiment on three LLMs: Qwen-2.5-7B-Instruct [50], Llama-3.1-8B-Instruct [13], InternLM-2.5-7B-Chat [8].

Tab. 1 presents the performance of FlowXpert and baselines across four troubleshooting scenarios on OpsFlowBench. The results underscore FlowXpert's superiority, attributed to its comprehensive and precise domain knowledge and its alignment with practical application requirements. (1) **Effectiveness of Expertise Sources**. The zero-shot approach heavily depends on LLMs' pretraining corpus, which proves inadequate for troubleshooting and leads to extremely low recall. Although precision is not so poor due to semantically relevant steps generated by LLMs, it often fails to reconstruct complete workflows. Among retrieval-based methods, VectorRAG prioritizes breadth, while GraphRAG emphasizes depth, yet neither fully capitalizes on both advantages. By integrating these knowledge bases and structuring them through a domain ontology, FlowXpert achieves a balance between knowledge breadth and precision. While broader knowledge introduces minor noise, slightly reducing precision, FlowXpert significantly improves recall by retrieving a more complete set of core steps, which OCEs prioritize in practice. (2) **Impact of Tuning Approaches**. CoT facilitates the knowledge application in a straightforward manner but remains limited and unstable. SFT enhances performance through language modeling on high-quality datasets, neglecting domain specificity. RL_GPT4o incorporates advanced model's feedback to refine workflow generation, while its unreliability may lead to adverse effects. In contrast, FlowXpert utilizes synthetic data to drive multi-agent coevolution, achieving performance comparable to or even surpassing that of SFT and RL_GPT4o. (3) **Generalizability Across Seed LLMs**. The results of different seed LLMs demonstrate that FlowXpert exhibits a degree of robustness and generalizability, suggesting its potential applicability across diverse model architectures.

Table 1: Overall performance across different scenarios on OpsFlowBench evaluated by STEPScore

Seed LLM	Method	STEPScore in Different Scenarios (%)														
		Hardware			Interface			Network			TOP			Average		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Qwen-2.5-7B-Instruct	zero-shot	76.4	72.3	73.7	70.1	67.2	68.0	**75.6**	69.5	71.9	66.4	60.0	62.5	71.6	66.8	68.5
	w/ VectorRAG	**78.1**	75.3	76.2	68.6	69.9	68.8	74.5	**75.6**	**74.6**	**67.9**	68.4	67.9	**72.2**	71.9	71.7
	w/ GraphRAG	73.8	77.0	74.9	70.1	70.8	70.1	65.3	65.8	64.9	65.8	67.9	66.3	69.3	71.2	69.8
	w/ CoT	76.6	76.7	76.4	**71.7**	73.2	**72.1**	68.7	73.1	70.5	64.9	67.4	65.8	70.7	72.5	71.2
	w/ SFT	67.5	70.5	68.5	65.7	70.5	67.5	63.2	68.6	65.3	61.6	66.2	63.3	64.6	68.8	66.2
	w/ RL_GPT4o	76.1	76.6	76.0	69.7	72.2	70.5	69.0	70.0	69.1	67.3	70.0	**68.2**	70.9	72.6	71.3
	FlowXpert (0th iteration)	74.8	78.1	76.0	70.2	71.7	70.7	70.0	73.0	71.0	63.8	66.0	64.5	69.6	72.1	70.4
	FlowXpert (1st iteration)	77.3	78.2	77.4	68.4	71.7	69.6	68.4	74.5	70.9	66.6	**70.4**	68.0	70.7	**73.8**	71.8
	FlowXpert (2nd iteration)	77.2	**78.3**	**77.5**	71.0	**73.3**	71.7	70.7	73.0	71.4	67.6	67.0	66.7	71.9	72.9	**71.9**
Llama-3.1-8B-Instruct	zero-shot	65.8	62.5	63.6	49.7	45.6	47.3	71.0	65.6	67.2	56.4	49.1	51.9	59.8	54.8	56.6
	w/ VectorRAG	75.2	74.7	74.6	70.6	67.8	68.6	69.5	70.8	69.7	63.9	63.5	63.2	69.8	69.0	69.0
	w/ GraphRAG	71.0	74.1	72.1	67.6	**70.2**	68.6	64.0	68.0	65.5	64.6	**66.7**	**65.3**	67.3	**70.1**	68.2
	w/ CoT	78.2	73.4	75.4	70.2	67.0	68.2	**72.4**	**74.8**	**73.1**	66.0	64.8	64.8	71.7	69.3	**70.0**
	w/ SFT	**79.6**	72.7	75.3	**71.4**	66.1	68.2	70.7	62.3	65.1	**69.0**	61.5	64.6	**73.2**	66.3	69.0
	w/ RL_GPT4o	77.8	72.8	74.7	71.0	66.4	68.1	69.9	72.5	70.6	66.0	63.3	64.2	71.4	68.2	69.3
	FlowXpert (0th iteration)	76.7	**75.6**	**75.7**	71.0	69.1	**69.5**	70.6	71.5	70.6	65.7	64.2	64.4	71.1	69.9	**70.0**
	FlowXpert (1st iteration)	77.0	72.8	74.4	69.7	68.2	68.6	71.4	71.9	71.1	65.5	63.9	64.1	70.9	68.8	69.3
	FlowXpert (2nd iteration)	74.8	71.9	72.3	70.7	66.5	68.1	69.8	70.5	69.7	62.4	59.2	60.3	69.2	66.4	67.3
InternLM-2.5-7B-Chat	zero-shot	74.0	72.4	72.4	69.3	67.9	67.9	71.9	65.6	67.3	67.2	59.3	62.5	70.5	66.3	67.5
	w/ VectorRAG	76.6	72.7	74.0	69.3	66.3	67.1	**77.2**	72.2	**74.0**	66.5	61.5	63.3	71.8	67.5	69.0
	w/ GraphRAG	71.4	75.6	72.7	71.4	69.8	69.9	70.8	66.8	67.9	64.9	64.8	64.5	69.2	69.7	68.8
	w/ CoT	75.0	73.3	73.5	**71.9**	67.9	69.2	70.6	**73.5**	71.3	65.3	60.7	61.7	70.6	68.0	68.4
	w/ SFT	**82.0**	**76.2**	**78.5**	70.7	68.0	68.9	71.6	71.6	71.1	**72.2**	65.5	**68.3**	**75.0**	70.3	**72.1**
	w/ RL_GPT4o	75.2	74.0	74.0	69.3	71.2	69.9	66.9	69.3	67.7	66.5	67.5	66.5	70.0	70.6	69.9
	FlowXpert (0th iteration)	72.5	75.9	73.5	66.7	**71.7**	68.8	66.3	72.0	68.6	64.3	65.5	64.4	67.8	71.1	68.9
	FlowXpert (1st iteration)	73.2	75.8	73.8	69.8	71.4	**70.3**	67.1	70.8	68.3	64.2	68.4	65.8	68.7	**71.8**	69.7
	FlowXpert (2nd iteration)	72.3	74.8	72.8	68.2	70.4	68.9	70.0	72.0	70.1	65.7	**69.3**	66.8	68.9	71.7	69.6

5.3 RQ3: Ablation Study

To validate the contribution of FlowXpert's core components, we conduct an ablation study under different conditions: A1: without knowledge base, A2: without graph base, A3: without vector base; B1: only fine-tune Planner, B2: only fine-tune Scorer. The results, as shown in Tab. 2, reveal two key findings: (1) **Importance of a Comprehensive Knowledge Base (A1, A2, A3)**. Removing certain knowledge sources reduces recall and workflow completeness. Although it may enhance precision by filtering out noise, OCEs prioritize full retrieval of core steps. Thus, optimizing for overall F1 score is more effective than solely maximizing precision. (2) **Effectiveness of Coevolution (B1, B2)**. Independently fine-tuning either Planner or Scorer yields improvements over FlowXpert's initial iteration. However, this approach constrains further performance gains that could be brought by coevolutionary learning.

Table 2: The evaluation results of ablation study

Seed LLM	Method	Average STEPScore (%)		
		Precision	Recall	F1
Qwen2.5-7B-Instruct	A1: w/o Knowledge Base	71.6	66.8	68.5
	A2: w/o Graph Base	**72.2**	71.9	71.7
	A3: w/o Vector Base	71.4	71.9	71.2
	B1: w/o DPO fine-tuning	70.3	72.9	71.2
	B2: w/o PPO fine-tuning	70.0	72.5	70.8
	FlowXpert (0th iteration)	69.6	72.1	70.4
	FlowXpert (1st iteration)	70.7	**73.8**	71.8
	FlowXpert (2nd iteration)	71.9	72.9	**71.9**

6 FlowXpert in Production

This section elaborates on how FlowXpert functions in a live production environment. We evaluate the quality of generated workflows through OCEs' usage in daily incident management (Sec. 6.1). Moreover, we perform a case study to further demonstrate the potential of AI Executors equipped with workflows for autonomous incident management (Sec. 6.2).

6.1 Online Deployment: For OCEs

Huawei Cloud's datacenter network (DCN) spans 17 regions and 63 availability zones, hosting $O(10^6)$ servers and $O(10^5)$ switches, and generating approximately 20,000 incidents monthly. To optimize OCEs' incident handling, these incidents are aggregated into a management system, Alarmagnify, where FlowXpert is integrated. The system operates on a high-performance Linux server equipped with an Intel(R) Xeon(R) Gold 6140 2.30GHz CPU and eight NVIDIA V100 GPUs, each with 32GB VRAM. Leveraging operation documents and 189 common incident queries from the DCN team, we perform a 2.2-hour knowledge base construction followed by a 15.1-hour coevolution, transforming naive LLMs (Qwen2.5-7B-Instruct) into specialized Planner and Scorer. For each ticket, FlowXpert generates a tailored troubleshooting workflow based on incident name and description, which OCEs can then use for further analysis.

Effectiveness. In the production environment, FlowXpert generates workflows for 189 common incident types. We calculate the STEPScore using manually curated workflows, achieving precision, recall, and F1 scores of 63.2, 78.4, and 69.6 respectively. These metrics demonstrate FlowXpert's capability to produce high-quality workflows. Additionally, OCEs use the generated workflows for troubleshooting analysis. A workflow is deemed acceptable by OCEs if it closely aligns with the standard incident-handling process, in quantitative terms, it recalls at least 75% of core steps. From

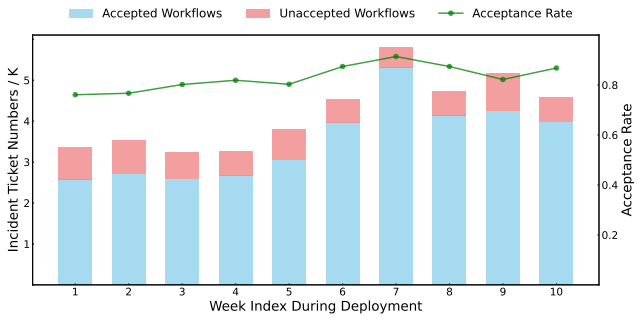

Figure 5: The weekly acceptance rate of workflows during the 10-week deployment

October 21 to December 29, 2024, we gathered data on 34,488 incident tickets, tracking the number of accepted workflows and the weekly acceptance rates. As shown in Fig. 5, approximately 80% of the workflows effectively guided OCEs step by step in incident management. These findings suggest that FlowXpert is capable of orchestrating high-quality workflows that are useful in real-world deployment.

Efficiency. As described in Sec. 3.2, developing a workflow for a single incident previously took a team of seven OCEs about seven hours, involving tasks such as identifying key metrics, assessing coverage and effectiveness, and designing and testing workflows. Notably, the team includes two experts whose expertise is indispensable but difficult to quantify temporally. With FlowXpert deployed, the time required to generate a workflow for each incident has been reduced to an average of 22.1 seconds, significantly reducing both labor and time costs. Intuitively, FlowXpert 's minute-level generation combined with rapid validation by a single OCE can, to some extent, replicate the 7-hour effort of a 7-person OCE team, including contributions from 2 experts.

6.2 Case Study: For AI Executors

Furthermore, we develop an AI Executor powered by Pangu-7B [37] to handle five categories of high-frequency incidents. As shown in Fig. 6, when an incident is triggered, FlowXpert organizes the troubleshooting workflow. After simple verification and refinement by OCEs, the Executor carries out each step sequentially: In "Process" steps, the Executor conducts intent recognition, parameter extraction, and tool invocation; In "Decision" steps, it performs logical reasoning and transition determination. The Executor integrates intermediate responses and delivers analysis results. This case illustrates that, following the workflow from the deployed FlowXpert, the autonomous Executor effectively carries out troubleshooting analysis in the production environment. Moreover, as indicated in Tab. 5 of Appendix C, the AI Executor enhances incident handling efficiency while minimizing interruptions to OCEs.

6.3 Lessons Learned

Three main threats challenge the validity of FlowXpert in deployment, and we try to suggest possible solutions:

Novel Incident Handling. For out-of-distribution incidents, FlowXpert retrieves relevant contexts from the knowledge base. Then

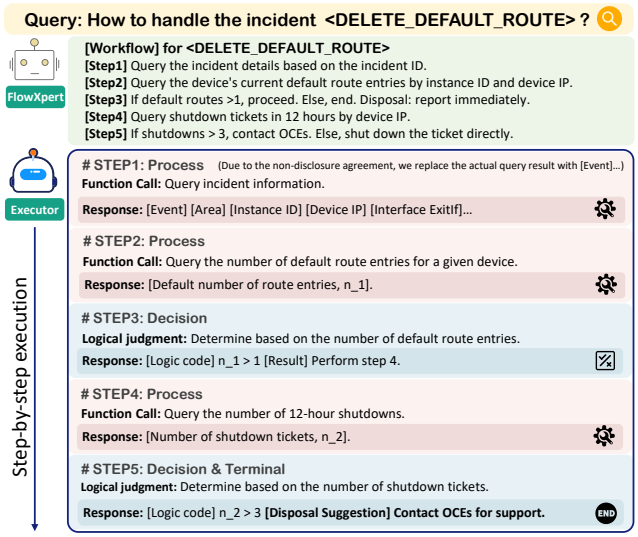

Figure 6: Autonomous AI Executor for incident handling

Planner orchestrates workflows by leveraging historical handling of similar cases, emulating experts' analogical reasoning. As for entirely novel incidents with no prior experience, manual handling followed by periodic updates to the knowledge base is a good choice, which requires only the addition of new chunks and nodes.

Execution Constraints. API sets are inadequate to fully capture troubleshooting expertise. Additionally, certain operations, such as physically checking if a fan blade is stuck, are hard to execute and assess in real time. Given these constraints, our workflow generation relies on step descriptions in natural language rather than fully executable APIs, potentially affecting the real-world executability. However, we validate FlowXpert's effectiveness in guiding execution within the real-world production (Sec. 6.1 and Sec. 6.2).

Coevolution Optimization. The effectiveness of coevolution depends on synthetic data quality. In Tab. 5.2, performance improves with additional iterations for Qwen and InternLM, but declines for Llama, which appears to have limited Chinese language comprehension. Therefore, we introduce consistency validation by Scorer. Also, human intervention in refining the synthetic data could enhance quality but requires a trade-off between performance and manual effort.

7 Conclusion

This work presents FlowXpert, an automated framework for troubleshooting workflow orchestration. Initially, we build a knowledge base incorporating vector and graph indexing, which leverages incident-aware nodes to sketch expertise precisely. Subsequently, reinforcement learning is applied to refine the workflow generator and evaluator, enabling multi-agent coevolution. Benchmark tests on the constructed OpsFlowBench, evaluated by the tailored STEP-Score metric, demonstrate FlowXpert's effectiveness. Additionally, real-world deployment highlights its contributions to OCEs and AI Executors. We believe that the concept of transforming naive LLMs into domain experts, through knowledge support and application enhancement, will benefit more areas beyond troubleshooting.

References

[1] 2014. Mermaid - Generation of diagrams like flowcharts or sequence diagrams from text in a similar manner as markdown. https://mermaid-js.github.io

[2] 2021. sentence-transformers/all-MiniLM-L6-v2. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. Accessed: 2024-08.

[3] 2023. Alibaba Cloud Health Dashboard. https://status.aliyun.com/#/historyEvent.

[4] 2023. Google Cloud Services Hit by Outage in Paris. https://thenewstack.io/google-cloud-services-hit-by-outage-in-paris/.

[5] 2023. Microsoft cloud outage hits users around the world. https://edition.cnn.com/2023/01/25/tech/microsoft-cloud-outage-worldwide-trnd/index.html.

[6] Kaikai An, Fangkai Yang, Junting Lu, Liqun Li, Zhixing Ren, Hao Huang, Lu Wang, Pu Zhao, Yu Kang, Hua Ding, et al. 2024. Nissist: An incident mitigation copilot based on troubleshooting guides. *arXiv preprint arXiv:2402.17531* (2024).

[7] Kathrin Blagec, Adriano Barbosa-Silva, Simon Ott, and Matthias Samwald. 2022. A curated, ontology-based, large-scale knowledge graph of artificial intelligence tasks and benchmarks. *Scientific Data* 9, 1 (2022), 322.

[8] Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, and et al. 2024. Internlm2 technical report. *arXiv preprint arXiv:2403.17297* (2024).

[9] Rong-Ching Chang and Jiawei Zhang. 2024. CommunityKG-RAG: Leveraging Community Structures in Knowledge Graphs for Advanced Retrieval-Augmented Generation in Fact-Checking. *arXiv preprint arXiv:2408.08535* (2024).

[10] Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, et al. 2024. T-eval: Evaluating the tool utilization capability of large language models step by step. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 9510–9529.

[11] Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. 2025. SFT Memorizes, RL Generalizes: A Comparative Study of Foundation Model Post-training. *arXiv preprint arXiv:2501.17161* (2025).

[12] R Du, H An, K Wang, and W Liu. 2024. A short review for ontology learning: Stride to large language models trend. *arXiv preprint arXiv:2404.14991* (2024).

[13] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* (2024).

[14] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130* (2024).

[15] Lisa Ehrlinger and Wolfram Wöß. 2016. Towards a definition of knowledge graphs. *SEMANTiCS (Posters, Demos, SuCCESS)* 48, 1-4 (2016), 2.

[16] Shengda Fan, Xin Cong, Yuepeng Fu, Zhong Zhang, Shuyan Zhang, Yuanwei Liu, Yesai Wu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024. WorkflowLLM: Enhancing Workflow Orchestration Capability of Large Language Models. *arXiv preprint arXiv:2411.05451* (2024).

[17] Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. 2024. GPTScore: Evaluate as You Desire. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. 6556–6576.

[18] Zorik Gekhman, Gal Yona, Roee Aharoni, Matan Eyal, Amir Feder, Roi Reichart, and Jonathan Herzig. 2024. Does Fine-Tuning LLMs on New Knowledge Encourage Hallucinations? *arXiv preprint arXiv:2405.05904* (2024).

[19] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models. *arXiv preprint arXiv:2405.14831* (2024).

[20] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

[21] Quzhe Huang, Mingxu Tao, Chen Zhang, Zhenwei An, Cong Jiang, Zhibin Chen, Zirui Wu, and Yansong Feng. 2023. Lawyer llama technical report. *arXiv preprint arXiv:2305.15062* (2023).

[22] Jiajun Jiang, Weihai Lu, Junjie Chen, Qingwei Lin, Pu Zhao, Yu Kang, Hongyu Zhang, Yingfei Xiong, Feng Gao, Zhangwei Xu, et al. 2020. How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1410–1420.

[23] Zhouyu Jiang, Mengshu Sun, Lei Liang, and Zhiqiang Zhang. 2024. Retrieve, summarize, plan: Advancing multi-hop question answering with an iterative approach. *arXiv preprint arXiv:2407.13101* (2024).

[24] Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Ren Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, et al. 2024. RLAIF vs. RLHF: Scaling Reinforcement Learning from Human Feedback with AI Feedback. In *Forty-first International Conference on Machine Learning*.

[25] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems* 33 (2020), 9459–9474.

[26] Zelong Li, Shuyuan Xu, Kai Mei, Wenyue Hua, Balaji Rama, Om Raheja, Hao Wang, He Zhu, and Yongfeng Zhang. 2024. Autoflow: Automated workflow generation for large language model agents. *arXiv preprint arXiv:2407.12821* (2024).

[27] Lei Liang, Mengshu Sun, Zhengke Gui, Zhongshu Zhu, Zhouyu Jiang, Ling Zhong, Yuan Qu, Peilong Zhao, Zhongpu Bo, Jin Yang, et al. 2024. Kag: Boosting llms in professional domains via knowledge augmented generation. *arXiv preprint arXiv:2409.13731* (2024).

[28] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[29] Yougang Lyu, Lingyong Yan, Shuaiqiang Wang, Haibo Shi, Dawei Yin, Pengjie Ren, Zhumin Chen, Maarten de Rijke, and Zhaochun Ren. 2024. KnowTuning: Knowledge-aware Fine-tuning for Large Language Models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. 14535–14556.

[30] Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, and Jian Guo. 2024. Think-on-graph 2.0: Deep and interpretable large language model reasoning with knowledge graph-guided retrieval. *arXiv e-prints* (2024), arXiv–2407.

[31] Nandana Mihindukulasooriya, Sanju Tiwari, Carlos F Enguix, and Kusum Lata. 2023. Text2kgbench: A benchmark for ontology-driven knowledge graph generation from text. In *International Semantic Web Conference*. Springer, 247–265.

[32] Harsha Nori, Yin Tat Lee, Sheng Zhang, Dean Carignan, Richard Edgar, Nicolo Fusi, Nicholas King, Jonathan Larson, Yuanzhi Li, Weishung Liu, et al. 2023. Can generalist foundation models outcompete special-purpose tuning? case study in medicine. *arXiv preprint arXiv:2311.16452* (2023).

[33] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.

[34] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *The Twelfth International Conference on Learning Representations*.

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

[36] N Reimers. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv preprint arXiv:1908.10084* (2019).

[37] Xiaozhe Ren, Pingyi Zhou, Xinfan Meng, Xinjing Huang, Yadao Wang, Weichao Wang, Pengfei Li, Xiaoda Zhang, Alexander Podolskiy, Grigory Arshinov, et al. 2023. Pangu-{\Sigma}: Towards trillion parameter language model with sparse heterogeneous computing. *arXiv preprint arXiv:2303.10845* (2023).

[38] Bhaskarjit Sarmah, Dhagash Mehta, Benika Hall, Rohan Rao, Sunil Patel, and Stefano Pasquali. 2024. Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction. In *Proceedings of the 5th ACM International Conference on AI in Finance*. 608–616.

[39] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[40] Manish Shetty, Chetan Bansal, Sai Pramod Upadhyayula, Arjun Radhakrishna, and Anurag Gupta. 2022. Autotsg: learning and synthesis for incident troubleshooting. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1477–1488.

[41] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509* (2022).

[42] Haopei Wang, Anubhavnidhi Abhashkumar, Changyu Lin, Tianrong Zhang, Xiaoming Gu, Ning Ma, Chang Wu, Songlin Liu, Wei Zhou, Yongbin Dong, et al. 2024. {NetAssistant}: Dialogue Based Network Diagnosis in Data Center Networks. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 2011–2024.

[43] Tianlu Wang, Ilia Kulikov, Olga Golovneva, Ping Yu, Weizhe Yuan, Jane Dwivedi-Yu, Richard Yuanzhe Pang, Maryam Fazel-Zarandi, Jason Weston, and Xian Li. 2024. Self-taught evaluators. *arXiv preprint arXiv: 2408.02666* (2024).

[44] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric Xing, and Zhiting Hu. 2024. PromptAgent: Strategic Planning with Language Models Enables Expert-level Prompt Optimization. In *The Twelfth International Conference on Learning Representations*.

[45] Yiying Wang, Xiaojing Li, Binzhu Wang, Yueyang Zhou, Yingru Lin, Han Ji, Hong Chen, Jinshi Zhang, Fei Yu, Zewei Zhao, et al. 2024. PEER: Expertizing Domain-Specific Tasks with a Multi-Agent Framework and Tuning Methods. *arXiv preprint arXiv:2407.06985* (2024).

[46] 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 (2022), 24824–24837.
[47] Alfred Ka Yiu Wong, Pradeep Ray, Nandan Parameswaran, and John Strassner. 2005. Ontology mapping for the interoperability problem in network management. *IEEE Journal on selected areas in Communications* 23, 10 (2005), 2058–2068.
[48] Tianhao Wu, Weizhe Yuan, Olga Golovneva, Jing Xu, Yuandong Tian, Jiantao Jiao, Jason Weston, and Sainbayar Sukhbaatar. 2024. Meta-rewarding language models: Self-improving alignment with llm-as-a-meta-judge. *arXiv preprint arXiv:2407.19594* (2024).
[49] Yiran Wu, Tianwei Yue, Shaokun Zhang, Chi Wang, and Qingyun Wu. 2024. StateFlow: Enhancing LLM Task-Solving through State-Driven Workflows. In *First Conference on Language Modeling*.
[50] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115* (2024).
[51] Rui Yang, Boming Yang, Aosong Feng, Sixun Ouyang, Moritz Blum, Tianwei She, Yuang Jiang, Freddy Lecue, Jinghui Lu, and Irene Li. 2024. Graphusion: A RAG Framework for Knowledge Graph Construction with a Global Perspective. *arXiv preprint arXiv:2410.17600* (2024).
[52] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*.
[53] Xunjian Yin, Baizhou Huang, and Xiaojun Wan. 2023. ALCUNA: Large Language Models Meet New Knowledge. *CoRR* abs/2310.14820 (2023).
[54] Zhen Zeng, William Watson, Nicole Cho, Saba Rahimi, Shayleen Reynolds, Tucker Balch, and Manuela Veloso. 2023. FlowMind: automatic workflow generation with LLMs. In *Proceedings of the Fourth ACM International Conference on AI in Finance*. 73–81.
[55] Tianyi Zhang*, Varsha Kishore*, Felix Wu*, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating Text Generation with BERT. In *International Conference on Learning Representations*.
[56] Tianjun Zhang, Shishir G Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, and Joseph E. Gonzalez. 2024. RAFT: Adapting Language Model to Domain Specific RAG. In *First Conference on Language Modeling*.
[57] Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*. Association for Computational Linguistics, Bangkok, Thailand.

A Implementation Details

We implement FlowXpert with Pytorch 2.4.1, CUDA 12.1, transformers 4.46.1, peft 0.12.0, trl 0.11.3, llamafactory 0.9.2 [57], neo4j 5.27.0, and langchain 0.3.2. And we utilize a popular Sentence-BERT [36] model, all-MiniLM-L6-v2 [2], as the embedding model for knowledge base construction, retrieval, and similarity calculation, *etc.* The benchmark tests are conducted on a high-performance Linux server with two Intel Xeon Gold 5416S CPUs and eight NVIDIA A6000 GPUs, each with 48GB of VRAM.

Dataset Split. We gather 252 data pairs, (query, workflow), across four distinct scenarios including Hardware, Interface, Network, and Top. First, we sort the data pairs in each scenario according to the workflow step count, to divide the data by task difficulty. The top 75% of the data pairs are labeled as "Hard", while the remaining pairs are classified as "Easy". Next, we partition the dataset for each difficulty level within each scenario. Through random sampling, 60% of the data pairs are allocated to the training set, with the remaining data pairs designated for the test set. The specific partitioning results are presented in Tab. 3, where each number represents the amount of data pairs in the dataset. Notably, the training process of FlowXpert only needs to utilize the queries from the training set, without the need for standard workflows.

Data Synthesis. In FlowXpert, we synthesize preference data for Direct Preference Optimization (DPO) [35]. For one iteration of the multi-agent coevolution, we generate three rounds of workflows for queries from the training set of OpsFlowBench. Each round produces three workflows of varying quality, based on the given context, which are then paired to create preference data. Finally, we obtain 1332 preference data pairs ($148 \times 3 \times 3$). The pairs are employed for DPO tuning after consistency validation by Scorer.

Table 3: Distribution of different scenarios

	Hardware	Interface	Network	Top	All
Train	49	33	18	48	148
Test	34	23	13	34	104

Hyperparameters. In practice, one iteration corresponds to one epoch of PPO and DPO fine-tuning for the Planner and Scorer, respectively. We show in Fig. 9 how the performance of FlowXpert varies with the number of iterations. Compared to the initial generation, FlowXpert improves the performance across different scenarios through fine-tuning. As the number of iterations increases, performance fluctuates but generally improves, indicating the contribution of coevolution. However, excessive iterations may lead to overfitting, causing performance degradation or slow convergence. Therefore, we typically select three iterations. In addition, we present the default value of main hyperparameters in Tab. 4.

Notably, we start the coevolution from seed LLMs rather than SFT models for two reasons. First, the performance of SFT on troubleshooting workflow generation is unstable as shown in Tab. 1. Second, open-source instruction-tuned models [8, 13, 50] already provide a strong initialization with a stable output format for workflow generation, which is typically a core goal of SFT stage [11].

Table 4: Descrptions of hyperparameters

Name	Description	Value
max_token	Maximum number of tokens the LLM can generate in the output sequence.	4096
temperature	Controls the randomness of LLM's output	1
DPO.batch_size	Batch size for DPO training.	4
PPO.batch_size	Batch size for PPO training.	4
DPO.learning_rate	Learning rate for DPO training.	5e-5
PPO.learning_rate	Learning rate for PPO training.	8e-6
lora_alpha	Scaling factor for rank decomposition in LoRA [20].	16
lora_rank	Rank of LoRA decomposition, defining the number of low-rank matrices.	8
lora_dropout	Dropout rate for LoRA layers.	0.05
N	Number of generated workflows per query in the online stage (Best-of-N).	3

B Prompt Design

We illustrate the prompt templates for graph base construction and multi-agent generation in Fig. 7 and Fig. 8, respectively.

Prompt Template for Extracting Incidents

System Prompt

```
# system role
You are a large language model skilled in extracting key information from documents.
Your task is to extract all possible incident_name entries from a document and provide a brief summary for
each incident_name. Each incident_name represents a specific event in the document.
# Instructions
1. When the user provides document content, first search for and extract all potential incident_name entries.
2. An incident_name is typically a phrase from the original document, and certain incident_name entries are
preceded by the string "Incident Case:".
3. For each incident_name, generate a brief summary (summarization) based on the document content.
4. Ensure the extracted incident_name and generated summarization are concise, clear, and accurately reflect
the information in the document.
5. If no incident_name is found in the document, return an empty list. Otherwise, return a list of tuples in the
format [(incident_name, summarization)].
Apply the above steps to process the document content provided by the user.
```

User Prompt

```
Refer to the following example to extract all incident_name entries (there may be none or multiple) from the
document. An incident_name is typically a phrase from the original document, and certain incident_name
entries are preceded by the string "Incident Case:".
Each incident_name represents a specific event in the document and requires a brief summary.
You must not return any additional content or explanation; only the generated tuple list [(incident_name,
summarization)] should be returned.
Example 1
Document Content:
"Incident Case: Fan module indicator remains red or flashes red
Phenomenon Description
The fan module indicator remains red or flashes red.
Related Alarms and Logs
Cause Analysis:
• The fan module is not fully inserted into the fan slot.
• The fan blade is stuck by foreign objects or too dusty, causing blockage.
• The fan module itself is faulty.
Steps:
1.Check if the fan module is properly inserted. The fan module supports hot-plugging; try reinserting the fan
module.
2.Remove the fan module and check if the fan blade is stuck by foreign objects or too dusty.
• If the fan blade is stuck, carefully remove the foreign object.
• If the fan blade is too dusty, clean it with a brush.
3. Replace the fan module with a working one of the same model. If the issue disappears, the fan module is
faulty and needs replacement.
2 Incident Case: Loud fan noise
Incident Case: Loud fan noise"
Returned Output:
[
("Fan module indicator remains red or flashes red", "The fan module indicator remaining red or flashing red
may be caused by the fan module not being properly inserted, the fan blade being stuck by foreign objects or
too dusty, or the fan module itself being faulty."),
("Loud fan noise", "")
]
Please extract the incident_name entries and their corresponding summarization from the following
document:
Document Content: <DOC>
```

Prompt Template for Filling in Nodes

System Prompt

```
# system role
You are a large language model specialized in completing tasks based on given prompts.
We use a Schema to formally define the Ontology of a specific domain while also providing a list of possible
incident_name entries from the document.
Your task is to generate JSON objects for all incident_name entries in the list based on the content of the
provided Document. These objects must match the predefined Schema and should be returned as a JSON
array.
# Instructions
1. Accept the incident_name list provided by the user. For each incident_name, extract relevant details from
the provided Document.
2. Ensure you fully understand the JSON Schema and its structure, including required fields, data types, and
constraints. Fill in the extracted information according to the predefined Schema and ignore irrelevant details.
3. Return a JSON array of objects that correspond exactly to the provided incident_name entries. Do not
modify the incident_name.
4. All responses must primarily be in Chinese, except for proper nouns.
```

User Prompt

```
This is a fill-in-the-blank task. We use a Schema to formally define the Ontology of a specific domain.
Referring to the following example, for each incident_name in the given list, extract relevant content from
the Document and fill it in detail into JSON objects that match the predefined Schema.
If no content is extracted for a field, its value must be set to None. Do not use null or any other value.
If the provided incident_name list is empty or the Document contains additional potential incident_name
entries not in the list, supplement the incident_name and corresponding JSON objects based on your
understanding of the Document.
You must not return any additional content or explanations; only return the list of generated JSON objects.
 # Example
<Incident>:
["Power Outage"]
Document:
 " In Community A, a major power outage occurred due to a sudden surge in electricity demand. The surge
caused circuit breakers to trip, as indicated by related alarms and records.
To resolve the issue, the main power source was restarted, the entire electrical system was inspected, and
faulty circuit breakers were replaced.
Such incidents are typically caused by extreme weather conditions and infrastructure issues.
Fortunately, backup generators were activated to maintain power continuity, and no reports of critical data
loss were made. "
[
  {
    "incident_name": "Power Outage",
    "failure_desc": "A power outage caused by a sudden surge in electricity demand. Related alarms include
circuit breaker trip logs.",
    "mitigation_steps": "Restarted the main power source, conducted a thorough inspection of the electrical
system, and replaced the faulty circuit breakers.",
    "typical_cases": None,
    "additional_info": "No reports of critical data loss; backup generators were activated."
  }
]
 # Schema: <SCHEMA>
 # Incident: <INCIDENT_NAMES>
 # Document: <DOC>
```

Prompt Template for Merging and Refining Nodes

System Prompt

```
# system role
You are a large language model specialized in text summarization.
Your task is to merge a list of dictionaries where each dictionary has the same keys and shares a common
primary key.
For each key, summarize the string values into a new, unified string value, ensuring no information is lost and
the resulting content is well-organized.
# Instructions
1. Accept a list of dictionaries where each dictionary has the same keys and a common primary key.
2. For each key, summarize its corresponding string values.
3. Ensure that no information is omitted, and the generated content is logically structured. If there are
repeated contents, rephrase or restructure them.
4. If all dictionaries have an empty value for a specific key, the value for that key must be None. Do not use
null or any other placeholder value.
5. Only return the single merged and summarized dictionary, without any additional content or explanations.
```

User Prompt

```
This is a task to merge multiple dictionaries into one.
Referring to the example below, merge all dictionaries in the given list into a single dictionary by summarizing
the string values for the same keys.
Ensure that no information is lost and that the content is logically organized. If content is repeated, rephrase
or restructure it. If all dictionaries have an empty value for a specific key, the value for that key must be None.
Do not use null or any other placeholder value.
You must not return any additional content or explanations; only return the single merged and summarized
dictionary.
# Example
<List>
[{
    "incident_name": "Database Connection Failure",
    "failure_desc": "Database connection timed out. Multiple connection attempts were unsuccessful.",
    "mitigation_steps": "1. Check if the database service is running. 2. Verify the network connection. 3.
Check if the firewall settings are blocking the database port.",
    "typical_cases": "Connection failure due to database service not running or network issues.",
    "additional_info": "Might be related to a system update."
  },
  {
    "incident_name": "Database Connection Failure",
    "failure_desc": ...
}]
<Response>
{
  "incident_name": "Database Connection Failure",
  "failure_desc": "Database connection timed out. Multiple connection attempts were unsuccessful.
Attempted to connect to the database via the client, but access was denied.",
  "mitigation_steps": "1.1. Check if the database service is running. 1.2. Verify the network connection. 1.3.
Check if the firewall settings are blocking the database port. 2.1. Check if user permissions are correct. 2.2.
Ensure the database server IP address has not been changed.",
  "typical_cases": "Connection failure due to database service not running or network issues. Connection
failure caused by insufficient user permissions or incorrect IP address configuration.",
  "additional_info": "Might be related to a system update."
}

# List: <NODE LIST>
```

Figure 7: Prompt templates for graph base construction

Prompt Template for Planner Generation

System Prompt

```
# system role
You are an intelligent assistant capable of generating workflows based on a given Context to answer user
Questions.
# Instructions
1. Understand the two types of Contexts provided: vector retrieval context and graph retrieval context.
2. Each type of context and its relevance to the Question decreases line by line (lines are separated by \n).
Analyze the content of the contexts, filtering out less relevant and later information in both types of contexts.
3. Based on the filtered content from both contexts, generate a workflow using Mermaid syntax.
4. Only return the workflow in Mermaid syntax, enclosed with special symbols $$ at the beginning and end.
```

User Prompt

```
The Context has two sources: vector retrieval and graph retrieval.
When referencing the Context, the relevance of each type of Context to the Question decreases line by line
(lines are separated by \n). It may be necessary to filter out less relevant and later information from each
Context.
Using the provided Context and your inner knowledge, generate a workflow to answer the Question,
following the example below.
Example:
$$
graph TD
A[Start] --> B[Check fan working status<br>Execute the command display device]
B --> C{{Is the fan status showing any issues?}}
C -->|Yes| D[Ensure the fan is properly connected<br>Check if the fan blades are stuck or dusty]
C -->|No| E[Problem resolved]
D --> F{{Has the fan status returned to normal?}}
F -->|Yes| G[Problem resolved]
F -->|No| H[Upgrade the fan software<br>Execute the command upgrade fan slot-id]
H --> I[Check if the fan status has recovered]
I -->|Yes| G[Problem resolved]
I -->|No| J[Replace the fan module or contact technical support]
$$
Question: <QUERY>
Context_by_vector: <CONTEXT_BY_VECTOR>
Context_by_graph: <CONTEXT_BY_GRAPH>
```

Prompt Template for Scorer Generation

System Prompt

```
# system role
You are a scorer capable of evaluating the workflow to solve a given Question, based on the provided
reference knowledge Context and the user's Question.
You will use the following scoring criteria: Relevance, Coverage, Accuracy, Coherence, and Conciseness.
Each scoring criterion has a range from 1 to 5, with 1 being the lowest score and 5 being the highest score.
```

User Prompt

```
Based on the given reference knowledge Context and the user's Question, score the workflow to solve the
Question.
The scoring criteria include five aspects:
1. Relevance: Whether the workflow is highly relevant to the given reference knowledge Context and reflects
the key points in the context, addressing the needs of the Question.
2. Coverage: Whether the workflow covers all necessary steps and conditions, ensuring the
comprehensiveness of the process.
3. Accuracy: Whether each step is accurate and effectively solves the problems or needs in the context.
4. Coherence: Whether the workflow is logically coherent, and the transitions between steps are natural and
reasonable.
5. Conciseness: Whether the workflow is clear, concise, and avoids unnecessary complexity or redundancy,
making it easy for the executor to understand and follow.

Question: <QUERY>
Context_by_vector: <CONTEXT_BY_VECTOR>
Context_by_graph: <CONTEXT_BY_GRAPH>
Workflow: <WORKFLOW>

Please score the above workflow.
Note! Only return a dictionary object in the format below. Do not add any content different from the format.
The format is as follows (X_i is an integer between 1 and 5, where 1 is the lowest score and 5 is the highest
score, and each X_i does not have to be the same):
$${{ "Relevance": X_1, "Coverage": X_2, "Accuracy": X_3, "Coherence": X_4, "Conciseness": X_5 }}$$
```

Figure 8: Prompt templates for Planner and Scorer generation

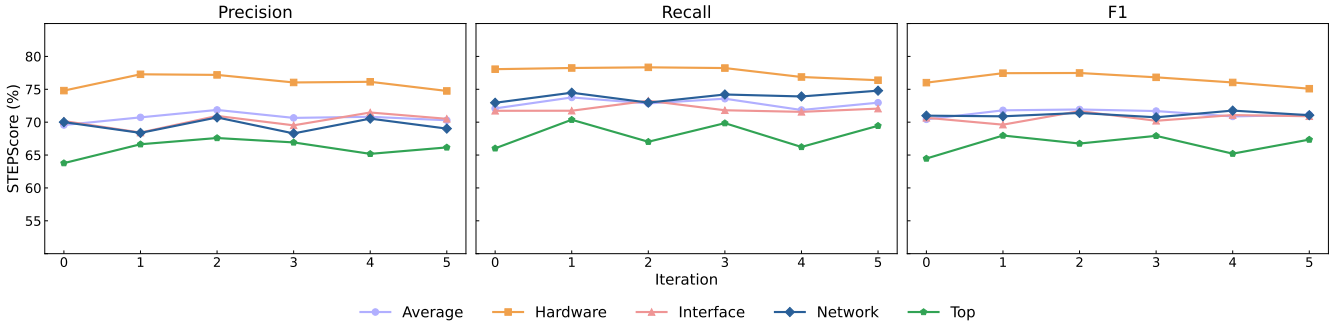

Figure 9: The effectiveness of FlowXpert (seed LLM: Qwen-2.5-7B-Instruct) under different iterations

Table 5: Human Time vs. Executor Time

Incident	Human Time(s)	Executor Time(s)
BGP_STATE_CHANGE_ESTABLISHED_TO_IDLE	373.4	200.9
BGP_BACKWARD_TRANSITION_ACTIVE	602.2	226.7
BGP_NOTIFICATION	801.8	232.0
DELETE_DEFAULT_ROUTE	166.4	111.5
NETWORK_DEVICE_OFFLINE_MONITOR	401.1	206.9

C Human Time vs. Executor Time

Tab. 5 presents five categories of high-frequency incidents, comparing the average handling time of human and AI Executor during deployment. The significant reduction in handling time highlights the efficiency of AI Executor.