Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.074384

ARTICLE

LEAF: A Lightweight Edge Agent Framework with Expert SLMs for the Industrial Internet of Things

Qingwen Yang[1], Zhi Li[2], Jiawei Tang[1], Yanyi Liu[1], Tiezheng Guo[1] and Yingyou Wen[1,*]

[1]School of Computer Science and Engineering, Northeastern University, Shenyang, 110169, China
[2]School of Information Science and Engineering, Shenyang Ligong University, Shenyang, 110159, China

*Corresponding Author: Yingyou Wen. Email: wenyingyou@mail.neu.edu.cn

Received: 10 October 2025; Accepted: 23 December 2025; Published: 12 March 2026

ABSTRACT: Deploying Large Language Model (LLM)-based agents in the Industrial Internet of Things (IIoT) presents significant challenges, including high latency from cloud-based APIs, data privacy concerns, and the infeasibility of deploying monolithic models on resource-constrained edge devices. While smaller models (SLMs) are suitable for edge deployment, they often lack the reasoning power for complex, multi-step tasks. To address these issues, this paper introduces LEAF, a Lightweight Edge Agent Framework designed for efficiently executing complex tasks at the edge. LEAF employs a novel architecture where multiple expert SLMs—specialized for planning, execution, and interaction—work in concert, decomposing complex problems into manageable sub-tasks. To mitigate the resource overhead of this multi-model approach, LEAF implements an efficient parameter-sharing scheme based on Scalable Low-Rank Adaptation (S-LoRA). We introduce a two-stage training strategy combining Supervised Fine-Tuning (SFT) and Group Relative Policy Optimization (GRPO) to significantly enhance each expert's capabilities. Furthermore, a Finite State Machine (FSM)-based decision engine orchestrates the workflow, uniquely balancing deterministic control with intelligent flexibility, making it ideal for industrial environments that demand both reliability and adaptability. Experiments across diverse IIoT scenarios demonstrate that LEAF significantly outperforms baseline methods in both task success rate and user satisfaction. Notably, our fine-tuned 4-billion-parameter model achieves a task success rate over 90% in complex IIoT scenarios, demonstrating LEAF's ability to deliver powerful and efficient autonomy at the industrial edge.

KEYWORDS: Industrial internet of things; edge computing; LLM-based agents; small language models

1 Introduction

The Industrial Internet of Things (IIoT) facilitates comprehensive perception and real-time monitoring of production processes through a vast network of sensors and devices. In such complex industrial settings, human operators struggle to master the extensive operational and maintenance details of all equipment. This reliance on expert personnel or lengthy manuals results in operational inefficiencies and a higher propensity for errors. Large Language Models (LLMs), such as GPT-4 and Claude, have shown remarkable capabilities in natural language understanding and generation, paving the way for fluent human-computer interaction. An LLM-based agent, using natural language as its interface, can comprehend user requirements from conversations and autonomously invoke external tools to execute tasks. This paradigm allows operators to manage complex equipment via simple commands, significantly lowering the operational barrier and boosting efficiency—a pivotal step in the ongoing intelligent transformation of IIoT.

Despite the promising applications of LLM-driven autonomous agents in IIoT, their practical deployment is fraught with challenges. Most existing agent frameworks depend on powerful, large-scale models, often with over one hundred billion parameters, making local deployment infeasible. The IIoT environment is characterized by network latency and bandwidth constraints, rendering remote API calls to cloud-based models inadequate for meeting real-time and stability requirements, while also introducing data privacy and security risks. Edge computing has emerged as a viable solution, offering low latency, high bandwidth, and data privacy. Advances in edge device hardware now permit the deployment of lightweight Small Language Models (SLMs). While these SLMs (typically under 10 billion parameters) offer rapid response times, their reasoning capabilities are comparatively limited, struggling with complex, multi-step tasks common in IIoT, such as long-chain fault diagnosis or multi-stage equipment control planning. Therefore, a critical challenge emerges: How to architect an agent framework that can be driven by resource-constrained SLMs to execute complex tasks efficiently at the edge?

It is widely recognized that an autonomous agent comprises core modules for role-playing, memory, planning, and action [1]. These modules demand a multi-faceted set of capabilities, including task planning, tool invocation, and dialogue generation, which are difficult to consolidate within a single, small-parameter model. To mitigate the complexity of planning, some research has proposed workflow-based methods, which reduce reliance on LLM reasoning. However, these workflows are often predefined and lack the flexibility and adaptability required for dynamic industrial environments. While techniques like fine-tuning and distillation can enhance SLM performance on specific tasks, a more comprehensive capability often requires the collaboration of multiple expert models. However, naively deploying multiple distinct models exacerbates the memory and computational burden, directly conflicting with the resource-constrained nature of edge devices.

To overcome these challenges, we propose LEAF, a Lightweight Edge Agent Framework designed for IIoT tasks. LEAF orchestrates multiple expert SLMs that work in concert, decomposing a complex task into discrete steps such as planning, execution, and interaction, while supporting dynamic decision-making based on the context. To enhance the proficiency of each expert, we employ a two-stage training strategy combining supervised fine-tuning and reinforcement learning. To specifically address the resource bottleneck imposed by the multi-expert approach, LEAF implements base parameter sharing via S-LoRA [2], drastically reducing the storage and computational overhead for edge deployment. Furthermore, to ensure operational stability and predictability—critical in industrial environments—we introduce a novel decision engine based on a Finite State Machine (FSM). This engine provides a deterministic control flow while leveraging the SLMs for dynamic decision-making at specific states, thereby balancing operational flexibility with procedural rigidity. Our experiments across multiple IIoT scenarios, including production lines, equipment maintenance, and energy management, demonstrate the superiority of LEAF in both task performance and token efficiency. Our main contributions can be summarized as follows:

- We propose a lightweight agent framework, LEAF, tailored for IIoT applications, enabling the efficient execution of complex agentic tasks on edge devices.
- We propose a novel training and collaboration mechanism for multiple expert SLMs, coupled with an S-LoRA-based parameter sharing scheme, which collectively enhances task-handling capabilities while resolving the resource constraints of edge deployment.
- We introduce a hybrid decision engine leveraging a FSM, which uniquely balances the need for operational flexibility and the demand for strict controllability inherent in IIoT applications.

2 Related Works

2.1 LLM-Based Agents

LLM-based agents understand natural language instructions and autonomously complete tasks by interacting with an environment. As LLM capabilities have advanced, agents have demonstrated remarkable potential in diverse domains, including coding and writing. The Chain-of-Thought (CoT) [3] method mimics human problem-solving by prompting an LLM to engage in step-by-step reasoning before providing an answer, thereby eliciting the model's decompositional abilities. ReAct [4] synergizes the reasoning of LLMs with interactions in external environments, effectively integrating "thought" and "action" to overcome the limitations of either mode alone. Models like ToolFormer [5] and ToolLLM [6] have been fine-tuned to enhance their ability to invoke external tools. Foundational frameworks like AgentVerse [7] and AutoGen [8] have streamlined agent development, while specialized platforms such as ChatDev [9] and MetaGPT [10] have optimized complex coding workflows, advancing capabilities in automated software development. Agent technology continues to mature rapidly, showing strong application potential across numerous fields.

2.2 Agents for IIoT

In the IIoT domains [11–13], agents find widespread application across various manufacturing stages, including production planning and scheduling, material handling, and fault diagnosis [14]. The integration of LLMs enhances traditional industrial agents, enabling them to better interpret human commands, decompose high-level ambiguous tasks into concrete executable steps, and orchestrate diverse IoT devices and AI modules. For instance, IoT-LLM [15] augments LLM perception and reasoning by incorporating sensor data and knowledge bases. IoT-Together [16] introduces a hybrid interaction model where the LLM collaborates with users to dynamically generate IoT services. For complex execution, the LLMind [17] framework translates natural language into control code to facilitate coordinated device operation. Addressing latency and privacy concerns, some research proposes edge-centric frameworks that deploy lightweight LLMs for direct, rapid control of IoT devices [18]. Despite these advancements, a critical gap remains: existing methods focus primarily on the deployment of lightweight models rather than augmenting their intrinsic reasoning capabilities. This limitation renders them insufficient for the complex, multi-step tasks required in IIoT scenarios.

3 Methodology

Fig. 1 shows the LEAF architecture, which employs three lightweight expert models (Planner, Executor, and Interactor) and a central Engine module to facilitate task execution and interaction. These models are trained in the cloud via a two-stage process to improve their planning, tool utilization, and interaction skills. For efficient edge deployment, the expert models are allowed to share base weights, thereby reducing demands on local storage and computation. Running on the edge also allows for continuous data collection, which is uploaded to the cloud for ongoing model optimization. The following sections detail the task formulation (Section 3.1), expert model design (Section 3.2), training strategies (Section 3.3), and the Engine module (Section 3.4).

3.1 Problem Formulation

The proliferation of IIoT has cultivated a complex, dynamic, and data-rich environment. Within this ecosystem, efficient and intuitive Human-Computer Interaction (HCI) is paramount for harnessing the full potential of interconnected industrial devices and systems. We introduce a lightweight agent framework, designed to serve as a natural language interface between human operators and the IIoT infrastructure.

The primary objective of this agentic system is to comprehend and correctly process user requests articulated in natural language, subsequently performing the appropriate actions, which may involve invoking specific tool APIs or generating informative responses. This moves beyond simple command-and-response systems towards a more proactive and goal-oriented dialogue paradigm.

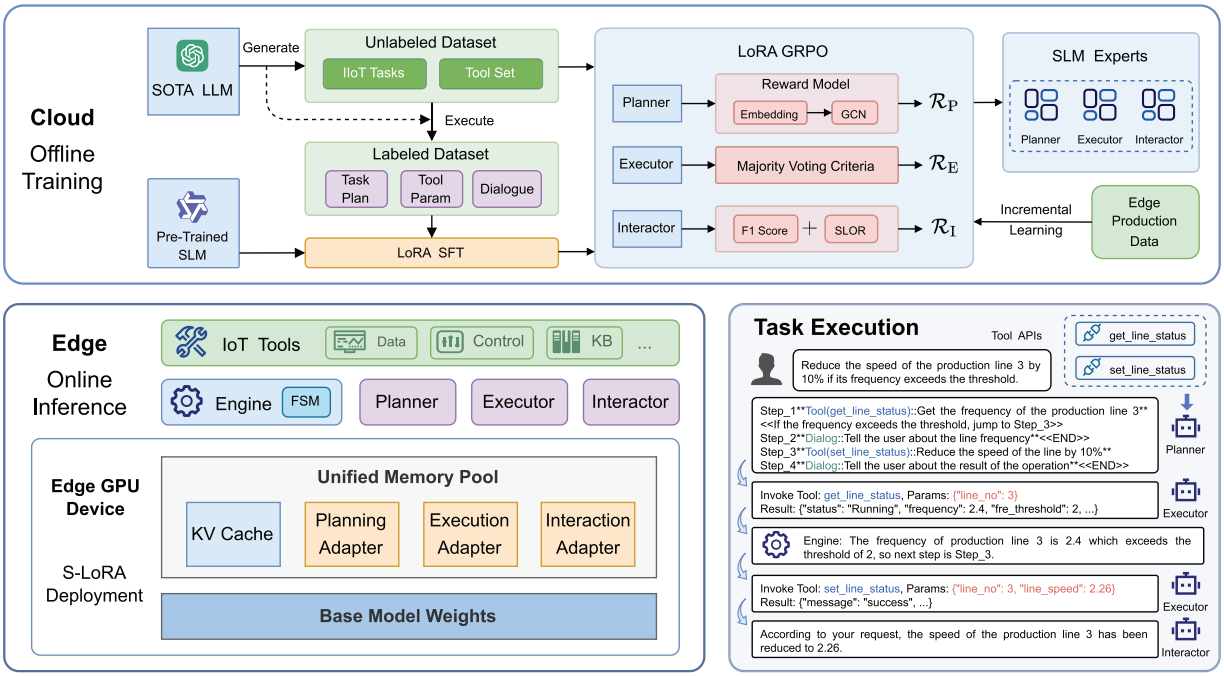


Figure 1: The overall architecture of LEAF and task execution example

Formally, we model this task as a sequential decision-making process. The agent's core function is to generate an optimal sequence of actions to fulfill a user's request, taking into account the available tools and the evolving conversational context. This process can be abstractly represented by the following high-level function:

$$\mathcal{C} = \text{LEAF}(r, \mathcal{T}, \mathcal{H}) \tag{1}$$

where:

- LEAF represents our proposed agent system, which embodies the policy for mapping inputs to a sequence of actions.
- r is the user's request, expressed as a natural language utterance, e.g., "What is the current operational status of the production line 3?" or "Monitor workshop energy consumption and alert if abnormal."
- $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_m\}$ is a finite set of available IIoT tools. Each tool τ_i corresponds to a specific API call that can interact with physical or virtual entities in the industrial environment (e.g., monitoring equipment state, adjusting production rate, or retrieving maintenance knowledge).
- $\mathcal{H}$ denotes the contextual information, which is critical for coherent interaction over multiple turns. This includes the dialogue history and tool execution results.
- $\mathcal{C} = \{a_1, a_2, \ldots, a_n\}$ is the dynamically constructed action sequence, representing the agent's behaviors to satisfy the request r.

The action a_i at each step is selected from a comprehensive action space $\mathcal{A}$, which is the union of the toolset $\mathcal{T}$ and a dialogue generation space $\mathcal{D}$:

$$a_i \in \mathcal{A} = \mathcal{T} \cup \mathcal{D} \tag{2}$$

Here, the dialogue generation space $\mathcal{D}$ encompasses all possible responses that serve various functions, such as asking clarifying questions, providing status updates, reporting the results of a tool execution, or delivering a final answer in a human-friendly manner.

Instead of learning a single, monolithic policy to generate the action sequence $\mathcal{C}$, we posit that the underlying decision-making process can be effectively decomposed into specialized functions. To this end, our LEAF framework operationalizes this task through the synergistic collaboration of three expert SLMs orchestrated by a central control unit. The ultimate objective is to enable the SLMs to dynamically construct an optimal action sequence $\mathcal{C}^*$ that ensures the correct tool usage, parameter passing, and fluent dialogue, thereby successfully and efficiently fulfilling the user's request.

3.2 Expert SLMs

To address the challenges of complex, multi-step IIoT tasks, LEAF employs a modular architecture instantiated from three expert SLMs: the Planner, the Executor, and the Interactor. Each is fine-tuned for a distinct role—planning, execution, and interaction, respectively—and designed for efficient deployment on edge devices. Through their synergistic collaboration, LEAF overcomes the functional limitations and reasoning bottlenecks inherent in monolithic SLM-based agents.

The entire workflow is orchestrated by the Planner, which serves as the strategic core of the framework. It is responsible for high-level task decomposition, interpreting the user's request from a global perspective to generate a structured execution plan. This marks a fundamental departure from prevailing paradigms like ReAct and ToolLLM, which demand intensive, step-by-step reasoning for dynamic action selection. Our Planner, in contrast, generates a comprehensive plan upfront, which is then interpreted by a deterministic Engine module. This architecture decouples high-level planning from low-level execution, significantly mitigating the real-time reasoning burden on the SLM. The plan generation process is formulated as:

$$\mathcal{S} = \mathrm{P}(r, \mathcal{T}, \mathcal{H}) \tag{3}$$

where $\mathcal{S} = \{s_1, s_2, \ldots, s_k\}$ is the execution plan. Each step s is defined as a tuple $s = (o, p, \tau, \kappa)$, where o is the action modality (either `Tool` or `Dialog`), p is a natural language instruction for this step, $\tau \in \mathcal{T}$ is the target tool to be invoked, and κ is a conditional flow control statement that dictates the transition logic (e.g., "if the tool output contains an error, jump to step 5"). Crucially, this plan is not a rigid sequence but a dynamic blueprint, empowering the Engine to adapt the execution path based on runtime outcomes.

The Executor is dedicated to action grounding, translating the abstract steps of the plan into concrete operations. For a `Tool` step, it parses the context $\mathcal{H}$ to extract the required arguments for the specified tool τ, formats them into a JSON object, and executes the API call. The Executor then enriches the context with the tool's output, making this new information available for subsequent steps and decisions.

The Interactor manages the human-computer dialogue channel. It generates context-aware, natural language responses to engage the user at various junctures—for example, to ask for clarification, report on progress, or deliver the final results. Its primary objective is to ensure a fluid and intuitive interactive experience, clearly and concisely communicating the agent's state and the task's outcome.

3.3 Training Strategy

Training expert SLMs directly on edge devices is infeasible due to significant computational constraints. We therefore train expert SLMs on the cloud, leveraging powerful computing resources to reduce training latency and minimize on-device overhead. The training of expert SLMs is conducted in two stages: Supervised Fine-Tuning (SFT) and Reinforcement Learning (RL), both of which utilize Low-Rank Adaptation (LoRA) [19] technology.

SFT is employed to enable the SLMs to learn the mapping between inputs and outputs, allowing them to comprehend task requirements and generate outputs that adhere to specifications. The SFT objective is tailored to the unique role of each expert:

- **Planner:** The goal is to produce syntactically correct and logically sound execution plans. We enforce a strict format for each plan step to ensure it is machine-parsable:
  `[No] **[Type] (tool) : : [Instr] **<<Ctrl>>`
  where `[No]` is the step index, `[Type]` is the action (`Tool` or `Dialog`), `(tool)` is the specified tool (omitted for `Dialog` steps), `[Instr]` contains the natural language instruction, and `Ctrl` is a flow control statement. The special token `END` within `Ctrl` demarcates the termination of the task.
- **Executor:** This model is trained to extract tool parameters from the dialogue context and output them as a clean, machine-readable JSON object, devoid of any irrelevant text.
- **Interactor:** Fine-tuning for the Interactor focuses on aligning its conversational style with the demands of the IIoT context. The objective is to generate responses that are not only accurate but also concise and clearly convey critical information.

Across all three experts, the SFT process is driven by minimizing a standard autoregressive, maximum likelihood objective. This is achieved via a cross-entropy loss over the target token sequence. The loss function is defined as:

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\sum_{i=1}^{|Y|} \log P(y_i | y_{<i}, X; \theta) \tag{4}$$

where θ represents the parameters of the SLMs; X is the input context, including the user request, dialogue history, and available tools; $Y = (y_1, y_2, \ldots, y_{|Y|})$ is the ground-truth target sequence (e.g., a formatted plan for the Planner, a JSON string for the Executor, or a natural language response for the Interactor).

To further align the expert models with the ultimate goal of effective task completion, we employ a second training stage using Reinforcement Learning. Specifically, we utilize Group Relative Policy Optimization (GRPO) [20], an algorithm that refines a policy by learning from ranked responses, which is particularly effective for complex generation tasks where defining a precise, absolute reward is challenging. The cornerstone of this RL stage is the design of a specialized reward function $\mathcal{R}$ for each expert model, tailored to its specific role within the framework.

The Planner's effectiveness depends on both the syntactic validity and logical rationality of its plan. To capture this, we combine a deterministic penalty for format errors with a learned reward model $\mathcal{R}_{\mathrm{RM}}$ that predicts plan quality. We parse the generated plan into a directed graph, where each step is a node and control flow statements define the edges between them. The text content of each node is then embedded and fed into a Graph Convolutional Network (GCN) to generate a holistic representation of the entire plan. Finally, a linear classifier is applied to the graph's aggregated representation to produce a quality score between 0 and 1. The final reward function for a generated plan $\mathcal{S}$ is defined as:

$$\mathcal{R}_{\mathrm{P}}(\mathcal{S}) = \begin{cases} \mathcal{R}_{\mathrm{RM}}(\mathcal{S}) & \text{if valid format and tools} \\ -1 & \text{otherwise} \end{cases} \tag{5}$$

This hybrid reward function strongly penalizes malformed plans while guiding the Planner towards generating strategies that are semantically coherent and likely to be effective.

For the Executor, correctness hinges on extracting the precise parameters for a given tool call. We devise a reward based on a majority voting principle. For a given input, we generate multiple candidate parameter sets. The consensus parameters are determined by the most frequently occurring value for each key. The reward for any single output is then proportional to its agreement with this consensus. The reward for an Executor output y_E (a JSON string) is formulated as:

$$\mathcal{R}_{\mathrm{E}}(y_E) = \begin{cases} \frac{1}{N_p} \sum_{j=1}^{N_p} \mathbb{I}(y_E[j] = v_j^*) & \text{if valid format} \\ -1 & \text{otherwise} \end{cases} \tag{6}$$

where N_p is the total number of parameters, v_j^* is the majority-voted value for the j-th parameter key, and $\mathbb{I}(\cdot)$ is the indicator function. This encourages the Executor to produce not only valid JSON but also the most plausible parameter values as determined by collective agreement.

A high-quality response from the Interactor must be both factually accurate and conversationally fluent. We design a composite reward function that balances these two aspects. Factual correctness is measured by extracting entities from the generated response and comparing them against the ground-truth entities in the context. Fluency is evaluated using the Syntactic Log-Odds Ratio (SLOR) [21]. The final reward for an Interactor response y_I is a weighted sum:

$$\mathcal{R}_{\mathrm{I}}(y_I) = \lambda_{\mathrm{fact}} \cdot \mathcal{R}_{\mathrm{fact}}(y_I) + \lambda_{\mathrm{fluency}} \cdot \mathcal{R}_{\mathrm{fluency}}(y_I) \tag{7}$$

where:

- $\mathcal{R}_{\mathrm{fact}}(y_I) = \mathrm{F1}(E_{\mathrm{gen}}, E_{\mathrm{ctx}})$, which is the F1-score comparing the set of entities extracted from the generated response (E_{gen}) with the set of ground-truth entities from the context (E_{ctx}).
- $\mathcal{R}_{\mathrm{fluency}}(y_I) = \mathrm{SLOR}(y_I)$, evaluates the grammatical coherence and naturalness of the response, which is calculated as:

$$\mathrm{SLOR}(y_I) = \frac{1}{L} \sum_{i=1}^{L} (\log P_{\mathrm{LM}}(w_i | w_{<i}) - \log P_{\mathrm{uni}}(w_i)) \tag{8}$$

  where $y_I = (w_1, \ldots, w_L)$ is the sequence of tokens of length L, $P_{\mathrm{LM}}(w_i | w_{<i})$ is the conditional probability of token w_i, and $P_{\mathrm{uni}}(w_i)$ is the independent unigram probability of the token.
- λ_{fact} and $\lambda_{\mathrm{fluency}}$ are hyperparameters that balance the trade-off between factual correctness and linguistic quality, set to 0.6 and 0.4, respectively.

This composite reward guides the Interactor to produce responses that are not only helpful and accurate but also natural and easy for the user to understand.

During the GRPO training phase, for each input context x, we generate a group of K candidate responses $\{y_1, y_2, \ldots, y_K\}$ from the current policy π_θ. We then use the expert-specific reward functions defined above to calculate a scalar reward $\mathcal{R}(y_k)$ for each candidate. Based on these reward scores, the responses are ranked from best to worst, forming a set of preference pairs (y_w, y_l), where y_w is preferred over y_l because $\mathcal{R}(y_w) > \mathcal{R}(y_l)$.

The policy π_θ is then updated by minimizing the GRPO loss, which is a form of pairwise ranking loss. The objective is to maximize the log-likelihood of the preferred responses over the less-preferred ones. The loss function is defined as:

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{O}}\left[\log\sigma\left(\beta\left(\log\pi_\theta(y_w|x) - \log\pi_\theta(y_l|x)\right)\right)\right] \tag{9}$$

where $\mathcal{O}$ is the dataset of preference pairs generated from the ranked responses, $\pi_\theta(y|x)$ is the likelihood of generating response y from the current policy, β is the temperature influencing how strongly the model should adhere to the preference margin, and $\sigma(\cdot)$ is the logistic sigmoid function.

In essence, this loss function encourages the policy π_θ to assign a higher probability to the preferred response (y_w) and a lower probability to the less-preferred one (y_l), progressively steering the model's generation behavior towards outputs that yield higher rewards.

3.4 Decision Engine

To govern the task execution workflow, we designed a Decision Engine based on the principles of a Finite State Machine (FSM), as illustrated in Fig. 2. This architecture codifies deterministic procedures into fixed state transitions while delegating complex, context-dependent choices to an SLM. This hybrid approach ensures both operational reliability and intelligent flexibility.

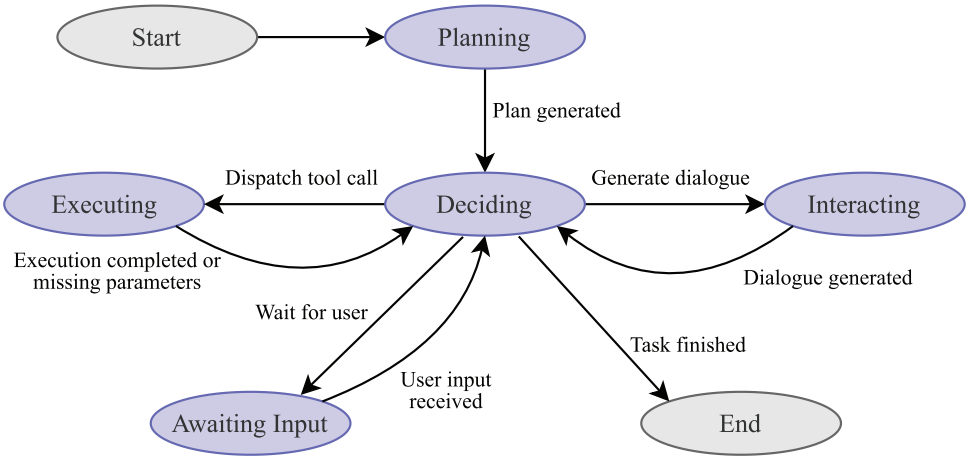


Figure 2: The FSM diagram of the Decision Engine. The central "Deciding" state orchestrates the flow between the "Executing" and "Interacting" states based on the plan and context

The FSM lifecycle begins in the **Start** state, transitioning to **Planning** where the Planner model generates an execution plan. Upon plan creation, the Engine enters the central **Deciding** state for the first time. The core operational loop of the Engine revolves around this state, which interprets the current plan step and context to select the subsequent state. The primary transitions are:

- **From Deciding:** The Engine uses an SLM to parse control statements and, based on the current task state, dispatches control to either the **Executing** state (for a tool call) or the **Interacting** state (for dialogue generation). If the plan is complete, it moves to the **End** state.
- **From Executing/Interacting:** Upon completion of their respective tasks (e.g., a tool call is finished or a dialogue is generated), these states transition back to **Deciding**, reporting the outcome. This allows the Engine to re-evaluate the context before proceeding.

- **Handling Missing Information:** A critical loop involves handling missing parameters. If the **Executing** state detects a missing parameter, it reports this failure to the **Deciding** state. The **Deciding** state's logic then routes the process to the **Interacting** state to request the missing data from the user. Once the user provides the input (**Awaiting Input** state), the process returns to the **Executing** state to retry the tool call.

This FSM design isolates dynamic, intelligent decision-making within the **Deciding** state, primarily for interpreting the plan's control statements (κ). All other transitions follow deterministic rules, significantly reducing the cognitive load on the SLM and ensuring a robust, predictable execution flow.

4 Experiments

4.1 Experimental Setup

To evaluate the performance of LEAF, we conducted experiments in three representative IIoT scenarios, each featuring 100 tasks and 20 tool APIs, designed to test distinct capabilities:

- **Production Line:** Focused on real-time monitoring and closed-loop control, involving tasks like line operation adjustment and production order replacement.
- **Equipment Maintenance:** Assessed the planning and execution of complete business processes, such as fault diagnosis and maintenance order generation.
- **Energy Management:** Tested automated resource management capabilities, with tasks including energy load optimization and cost analysis.

Data Generation

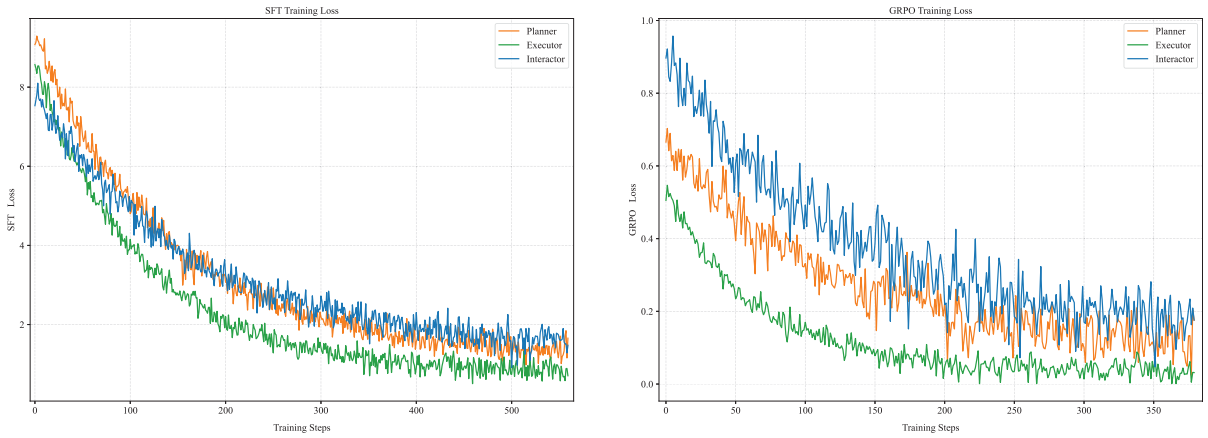
For each task, we generated diverse user requests using `gpt-4.1` with structured prompts. All samples were manually filtered for quality, yielding approximately 6k samples for SFT and 3k for GRPO. The training and test sets were strictly separated to prevent data leakage. While synthetic data has limitations, this method created a large, controlled dataset crucial for robust training and evaluation. Future work will focus on validating our framework with real-world data.

Baselines

We selected 5 methods as baselines, including 2 general agent methods (ReAct, ToolLLM) and 3 agent methods designed for IoT (ChatIoT, LLMind, IoT-Together). ChatIoT is a zero-code system based on large language models that automatically translates users' natural language requests into executable IoT trigger-action programs. LLMind is an agent framework for IoT tasks that uses the LLM as a commander to coordinate specialized AI modules and IoT devices. IoT-Together is a hybrid active interaction paradigm driven by user needs for code generation to create dynamic IoT services.

Implementation Details

LEAF employs `qwen3-4b` as the base model. For SFT training, the batch size is 32, trained for 3 epochs, with a learning rate ranging from 2e–5 to 5e–5. For GRPO training, β is set to 0.3 and the batch size is 16, trained for 2 epochs, with a learning rate ranging from 1e–6 to 5e–6. In both SFT and GRPO stages, we fine tuned all linear layer parameters of the model, with a LoRA rank of 8. The training losses of the three expert models are illustrated in Fig. 3. To ensure a fair comparison, all methods were implemented using `qwen3-4b` as the backbone model. LEAF was deployed using the vllm framework (v0.10.0) with a multi-adapter startup method. To specifically evaluate its performance on a representative edge device, the efficiency analysis in Section 4.3 was conducted on an NVIDIA Jetson AGX Orin (32 GB), while other experiments were conducted on an NVIDIA RTX 3090. For all experiments, the model's temperature was set to 0 to ensure deterministic and stable outputs.

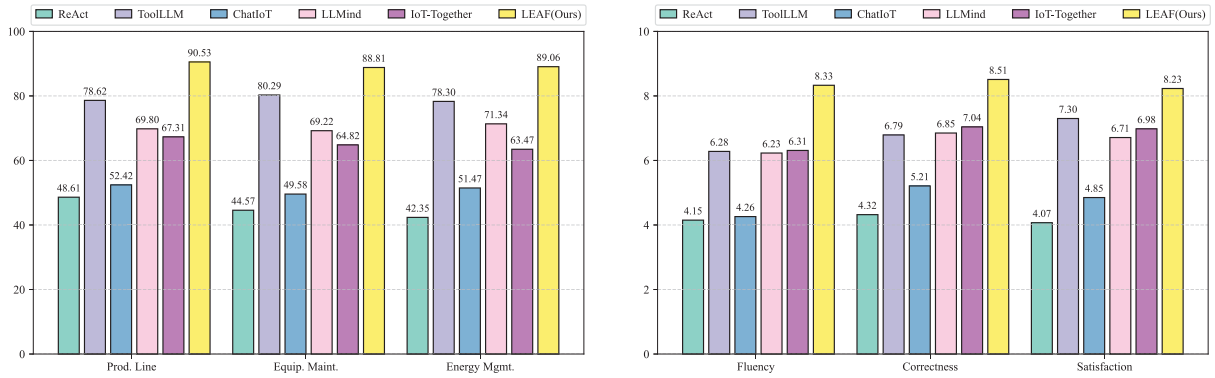


(**a**) SFT Loss. (**b**) GRPO Loss.

Figure 3: Loss curves for SFT and GRPO training

4.2 Main Results

We used both automatic evaluation and human evaluation to assess the performance of each method. Automatic evaluation involved conducting experiments on the test data from the three scenarios, with `gpt-4.1` evaluating the task success rate of each method. For human evaluation, We recruited 10 participants (5 with a computer science background and 5 with experience in industrial automation) to act as system users. They were given high-level task objectives within the production line scenario and encouraged to interact freely with the system. Afterward, they rated each method on a 10-point scale across three dimensions: fluency, correctness, and satisfaction. The experimental results are shown in Fig. 4.



(**a**) Task success rate. (**b**) Human evaluation results.

Figure 4: Main experimental results comparing LEAF with baseline methods across three IoT scenarios

As shown in Fig. 4a, LEAF achieved the highest task success rate in all scenarios: 90.53% in production line, 88.81% in equipment maintenance, and 89.06% in energy management. This indicates that the collaborative approach of multiple expert models in LEAF significantly improves task execution performance for edge agents. In contrast, the other methods exhibited lower success rates, as they rely solely on the intrinsic capabilities of a single lightweight model, which struggles with complex tasks. ToolLLM, having been fine-tuned for tool use, outperformed non-fine-tuned methods but was still limited by the single-model architecture in complex scenarios.

The results, presented in Fig. 4b, show that LEAF scored the highest in all three metrics. This suggests that LEAF not only completes tasks effectively but also provides a superior user experience. This is attributed to the synergistic design of its expert models: the Interactor ensures fluent and natural conversation, while the Planner and Executor enhance task correctness and efficiency, collectively boosting user satisfaction.

To validate generalization and ensure our framework avoids overfitting to the teacher model, we evaluated LEAF on two distinct test sets generated by `gpt-4.1` and a different SOTA model, `claude-4-opus`. We analyzed performance using fine-grained metrics: **Tool** (the proportion of correctly selected tools), **Execute** (the proportion of correct parameters in tool calls), and **Inform** (the proportion of correct information in the final dialogue). As shown in Table 1, LEAF substantially boosts the performance of `qwen3-4b` on the original test set. Crucially, despite a slight, expected drop from the distributional shift, it maintains a high task success rate of 88.72% on the new `claude-4-opus` set. This robust cross-model performance strongly indicates that our training methodology, particularly GRPO with tailored rewards, imparts generalizable problem-solving logic rather than simply mimicking stylistic patterns, thus validating our approach's effectiveness.

Table 1: Performance comparison across models on test sets generated by different SOTA LLMs

Model	Test Set: Generated by GPT-4.1				Test Set: Generated by Claude 4 Opus			
	Tool	Execute	Inform	Task SR.	Tool	Execute	Inform	Task SR.
qwen3-4b	68.16	73.25	75.08	67.43	67.55	73.89	74.31	66.95
qwen3-32b	84.79	89.93	90.24	82.50	83.12	88.54	90.43	81.68
qwen3-max	94.06	96.40	98.31	92.74	94.57	96.88	97.85	93.05
gpt-4.1	95.83	97.45	98.02	94.62	95.01	96.93	97.54	93.89
LEAF (qwen3-4b)	92.81	95.06	96.74	90.53	91.26	94.85	96.02	89.72

4.3 Efficiency Analysis

Fig. 5 illustrates the comparative token consumption of LEAF and the baseline methods across three distinct scenarios. Notably, LEAF consistently demonstrates lower token usage compared to all baselines. Methods such as ReAct and ToolLLM incur legitimately high token costs due to the requirement of providing the complete context at every step. Similarly, other baselines exhibit high consumption stemming from the use of LLMs for complex context analysis or code generation. LEAF addresses this by decomposing tasks among collaborative expert models. While the Planner's generated plans may increase in length, the aggregate token consumption is significantly reduced because the downstream Executor and Interactor utilize only simplified, task-relevant contexts.

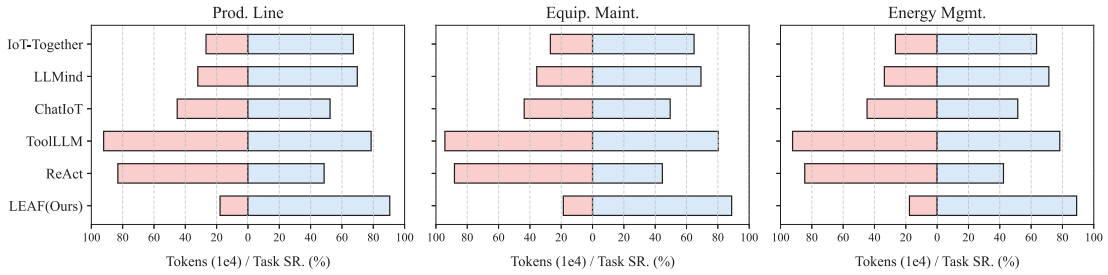


Figure 5: Visualization of the performance and the token consumption of different methods

To analyze the operational efficiency of LEAF, we measured the token consumption and average task time of each module across the three scenarios. As input tokens consist mainly of context and tool descriptions with minimal impact on latency, our analysis focused on output tokens. Table 2 shows that the total output tokens per scenario were consistent, ranging from 2.6×10^4 to 3×10^4. The Planner was the most token-intensive module, accounting for approximately 70% of the output, as it generates detailed execution plans.

Table 2: Token consumption per component across scenarios

Scenario	Token consumption				
	Planner	**Executor**	**Interactor**	**Engine**	**Total**
Prod. Line	21,364	2467	3762	2217	29,810
Equip. Maint.	19,859	2283	3154	2202	27,498
Energy Mgmt.	20,743	2754	3208	2354	29,059

To validate LEAF's suitability for deployment on resource-constrained edge hardware, we analyzed its computational latency and memory usage on an Jetson AGX Orin (32 GB) platform, as shown in Table 3. The reported time measures the framework's computational latency, excluding external network and tool API latencies. Across all scenarios, the average task completion time is under 5 s, with a peak VRAM consumption of 15.8 GB. These results confirm that LEAF's multi-model architecture operates efficiently within the memory and latency constraints of real-world industrial edge applications.

Table 3: Average time and peak memory consumption on an NVIDIA Jetson AGX Orin (32 GB)

Scenario	Average time consumption (ms)					Peak VRAM (GB)
	Planner	**Executor**	**Interactor**	**Engine**	**Total**	
Prod. Line	3445	387	608	370	4810	15.8
Equip. Maint.	3422	341	579	365	4707	15.5
Energy Mgmt.	3376	429	573	412	4790	15.7

4.4 Ablation Study

We conducted ablation studies to validate the contribution of LEAF's key design components. The settings were as follows: (1) **w/o FT:** Used the pre-trained SLM directly without any fine-tuning. (2) **w/o GRPO:** Used only SFT for training, omitting the GRPO stage. (3) **w/o Engine:** Removed the decision engine, forcing the framework to follow a fixed execution sequence. (4) **single model:** A single SLM was fine-tuned on the combined data of all expert roles to act as a monolithic agent baseline.

The results in Table 4 demonstrate that all three components are crucial for LEAF's performance. Without fine-tuning, the task success rate dropped significantly, highlighting that specialized fine-tuning is essential for unlocking the capabilities of SLMs for complex tasks. While SFT provided a significant performance boost, the removal of GRPO led to a noticeable decline, confirming that GRPO further refines the model's generation quality and execution effectiveness. Without using the Engine, the success rate also decreased, underscoring the Engine's vital role in dynamically controlling the task flow. For instance, when faced with an ambiguous user request like "check the equipment", the full LEAF framework correctly identifies the missing equipment ID and uses the Interactor to ask for clarification. The "w/o Engine" variant,

lacking this dynamic control, might proceed with a faulty tool call, leading to task failure. This underscores the Engine's vital role in dynamic and robust task flow management. In addition, LEAF consistently outperformed the monolithic single-model agent, justifying the necessity of its multi-expert architecture.

Table 4: Task success rate comparison across variants

Variant	Prod. Line	Equip. Maint.	Energy Mgmt.
LEAF	90.53	88.81	89.06
w/o FT	67.43	66.48	66.87
w/o GRPO	84.02	81.57	83.40
w/o Engine	88.31	86.14	86.97
Single Model	87.23	85.46	85.91

5 Conclusion

This paper introduces LEAF, a lightweight agent framework designed to enable complex task execution on resource-constrained edge devices in IIoT environments. LEAF utilizes a synergistic approach where multiple expert SLMs collaborate under the governance of a FSM-based Decision Engine. Our experiments across three IIoT scenarios demonstrate that LEAF significantly outperforms existing methods in both task success rate and user satisfaction, demonstrating its ability to deliver robust task execution on SLMs and confirming its suitability for deployment at the industrial edge. Future work will explore mechanisms for on-device model adaptation, allowing LEAF to continuously learn from new interactions and changing industrial environments, further enhancing its autonomy and scalability. Additionally, we plan to investigate automated tool discovery to broaden the agent's capabilities with minimal human intervention.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: Qingwen Yang: Conceptualization, methodology and writing—original draft preparation; Zhi Li: Software and validation; Jiawei Tang: Investigation and conceptualization; Yanyi Liu: Data curation and formal analysis; Tiezheng Guo: writing—review and editing; Yingyou Wen: Supervision and project administration. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The raw data supporting the conclusions of this article will be made available by the authors on reasonable request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Wang L, Ma C, Feng X, Zhang Z, Yang H, Zhang J, et al. A survey on large language model based autonomous agents. Front Comput Sci. 2024;18:186345. doi:10.1007/s11704-024-40231-1.
2. Sheng Y, Cao S, Li D, Hooper C, Lee N, Yang S, et al. S-LoRA: serving thousands of concurrent LoRA adapters. In: Gibbons P, Pekhimenko G, Sa CD, editors. Proceedings of the 5th MLSys conference. Vol. 6. Santa Clara, CA, USA: MLSys; 2024. p. 296–311.

3. Wei J, Wang X, Schuurmans D, Bosma M, Ichter B, Xia F, et al. Chain-of-thought prompting elicits reasoning in large language models. In: Advances in neural information processing systems. Vol. 35. Red Hook, NY, USA: Curran Associates, Inc.; 2022. p. 24824–37.
4. Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan KR, et al. ReAct: synergizing reasoning and acting in language models [Internet]. 2023 [cited 2025 Dec 2]. Available from: https://openreview.net/forum?id=WE_vluYUL-X
5. Schick T, Dwivedi-Yu J, Dessí R, Raileanu R, Lomeli M, Hambro E, et al. Toolformer: language models can teach themselves to use tools. In: Proceedings of the 37th international conference on neural information processing systems. Red Hook, NY, USA: Curran Associates, Inc.; 2023. p. 1–13.
6. Qin Y, Liang S, Ye Y, Zhu K, Yan L, Lu Y, et al. ToolLLM: facilitating large language models to master 16000+ real-world APIs [Internet]. 2024 [cited 2025 Dec 2]. Available from: https://openreview.net/forum?id=dHng2O0Jjr.
7. Chen W, Su Y, Zuo J, Yang C, Yuan C, Chan CM, et al. AgentVerse: facilitating multi-agent collaboration and exploring emergent behaviors [Internet]. 2024 [cited 2025 Dec 2]. Available from: https://openreview.net/forum?id=EHg5GDnyq1.
8. Wu Q, Bansal G, Zhang J, Wu Y, Li B, Zhu E, et al. AutoGen: enabling next-gen LLM applications via multi-agent conversations [Internet]. 2024 [cited 2025 Dec 2]. Available from: https://openreview.net/forum?id=BAakY1hNKS.
9. Qian C, Liu W, Liu H, Chen N, Dang Y, Li J, et al. ChatDev: communicative agents for software development. In: Proceedings of the 62nd annual meeting of the association for computational linguistics. Stroudsburg, PA, USA: Association for Computational Linguistics; 2024. p. 15174–86.
10. Hong S, Zhuge M, Chen J, Zheng X, Cheng Y, Wang J, et al. MetaGPT: meta programming for a multi-agent collaborative framework [Internet]. 2024 [cited 2025 Dec 2]. Available from: https://openreview.net/forum?id=VtmBAGCN7o.
11. Ma L, Li N, Guo Y, Wang X, Yang S, Huang M, et al. Learning to optimize: reference vector reinforcement learning adaption to constrained many-objective optimization of industrial copper burdening system. IEEE Trans Cybernet. 2022;52(12):12698–711. doi:10.1109/tcyb.2021.3086501.
12. Ma L, Huang M, Yang S, Wang R, Wang X. An adaptive localized decision variable analysis approach to large-scale multiobjective and many-objective optimization. IEEE Trans Cybern. 2022;52(7):6684–96. doi:10.1109/tcyb.2020.3041212.
13. Ma L, Wang X, Wang X, Wang L, Shi Y, Huang M. TCDA: truthful combinatorial double auctions for mobile edge computing in industrial internet of things. IEEE Trans Mobile Comput. 2022;21(11):4125–38. doi:10.1109/tmc.2021.3064314.
14. Kök İ., Demirci O, Özdemir S. When IoT meet LLMs: applications and challenges. In: 2024 IEEE international conference on big data (BigData). Piscataway, NJ, USA: IEEE; 2024. p. 7075–84.
15. An T, Zhou Y, Zou H, Yang J. IoT-LLM: enhancing real-world IoT task reasoning with large language models. arXiv:2410.02429. 2024.
16. Adnan B, Miryala S, Sambu A, Vaidhyanathan K, De Sanctis M, Spalazzese R. Leveraging LLMs for dynamic IoT systems generation through mixed-initiative interaction. In: 2025 IEEE 22nd international conference on software architecture companion (ICSA-C). Piscataway, NJ, USA: IEEE; 2025. p. 488–97.
17. Cui H, Du Y, Yang Q, Shao Y, Liew SC. LLMind: orchestrating AI and IoT with LLM for complex task execution. IEEE Commun Mag. 2025;63(4):214–20. doi:10.1109/mcom.002.2400106.
18. Chen H, Deng W, Yang S, Xu J, Jiang Z, Ngai ECH, et al. Toward edge general intelligence via large language models: opportunities and challenges. IEEE Netw. 2025;39(5):263–71. doi:10.1109/mnet.2025.3539338.
19. Hu EJ, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, et al. LoRA: low-rank adaptation of large language models [Internet]. 2022 [cited 2025 Dec 2]. Available from: https://openreview.net/forum?id=nZeVKeeFYf9.

20. Shao Z, Wang P, Zhu Q, Xu R, Song J, Bi X, et al. DeepSeekMath: pushing the limits of mathematical reasoning in open language models. arXiv:2402.03300. 2024.
21. Kann K, Rothe S, Filippova K. Sentence-level fluency evaluation: references help, but can be spared!. In: Korhonen A, Titov I, editors. Proceedings of the 22nd conference on computational natural language learning. Stroudsburg, PA, USA: Association for Computational Linguistics; 2018. p. 313–23.