P-LAM: Two-Step Reasoning for Proactive Large Action Models

Anonymous ACL submission

Abstract

Recent work has expanded the focus of Large Language Models (LLMs) to Large Action Models (LAMs), which generate executable actions for tasks such as coding, reasoning, and data analysis. However, existing LAMs typically remain reactive and cannot initiate actions without explicit user instructions, limiting their usefulness in dynamic settings. We introduce P-LAM, a two-step proactive reasoning framework that autonomously determines when intervention is required and what task to perform based on environmental observations. The first step evaluates the need for intervention and infers the task type, while the second applies a task-aligned prompt template to generate an appropriate action plan. Without any post-training, P-LAM achieves a 61.31% success rate on the contamination-resistant LiveBench benchmark, substantially outperforming a conventional Chain-of-Thought approach by 26.31% across six domains, including coding and writing. These results demonstrate that lightweight proactive reasoning can markedly improve LAM performance and reliability, even in zero-shot settings.

1 Introduction

Recent advances have shifted research focus from Large Language Models (LLMs) (Brown et al., 2020), primarily used for text generation, to Large Action Models (LAMs), which generate executable actions for complex tasks (Wang et al., 2024a; Piccoli et al., 2025). This transition expands the role of large models beyond language understanding and enables their application to domains such as coding, data analysis, and multi-step reasoning (Olausson et al., 2023; Yao et al., 2023b; Wang et al., 2023; Patil et al., 2024).

Traditional LAMs rely on elaborate multi-stage training pipelines involving extensive task-specific data collection, repeated expert demonstrations, and fine-tuning through supervised or reinforcement learning (Wang et al., 2024a; Zhang et al., 2025). Building such systems demands substantial engineering effort, including the construction of domain-specific datasets and reward functions and the careful adaptation of models to each target environment. Consequently, these models tend to become highly specialized and exhibit limited flexibility outside their training contexts.

Most existing systems further operate under an instruction-driven paradigm, initiating actions only in response to predefined prompts or explicit user inputs (Fan et al., 2022; Liu et al., 2023; Guo et al., 2024). This reactive behavior is insufficient in dynamic or open-ended scenarios where user guidance is unavailable and timely proactive intervention is required.

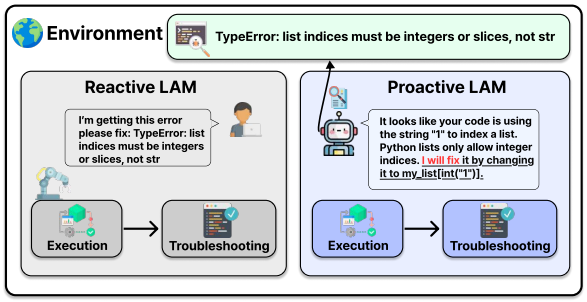

Figure 1: Illustration of bug handling: Reactive LAM waits for explicit user instructions before troubleshooting, whereas Proactive LAM captures errors from the environment and initiates troubleshooting prior to explicit user guidance.

We propose the **Proactive Large Action Model (P-LAM)**, a prompting-based framework that enables proactive task execution in zero-shot environments. P-LAM follows a two-step process: the first step determines whether intervention is necessary and identifies the underlying task type; the second step applies a task-aligned template to generate an appropriate action plan, supporting zero-shot generalization across diverse task distributions. Figure 1

illustrates how P-LAM detects and resolves errors before explicit user input.

We evaluate P-LAM on the contamination-resistant LiveBench benchmark (White et al., 2024), which continuously introduces novel tasks and poses substantial challenges even for state-of-the-art commercial models. All performance gains arise solely from the proposed two-step prompting strategy, without any post-training or additional instruction examples.

The primary contributions of this work are as follows:

- We introduce P-LAM, a proactive two-step reasoning framework that generates and executes actions directly from observations, without requiring explicit user instructions or curated examples.
- We propose a confidence-aware intervention mechanism that reduces unnecessary or incorrect proactive actions, improving overall reliability.
- We demonstrate through comprehensive evaluation on LiveBench that P-LAM enables smaller models to achieve competitive or even superior performance compared to significantly larger LAMs.

These contributions establish a foundation for developing proactive and generalizable LAMs, moving the field toward open-ended, real-world decision-making.

2 Related Work

This section reviews prior research on LAMs, proactive assistance frameworks, and prompt-based reasoning methods, and highlights the remaining gaps that motivate our task-agnostic proactive approach.

Reactive Large Action Models. Recent advances in language models have enabled the development of LAMs for domains such as code generation (Yang et al., 2024; Zhang et al., 2024a; Wang et al., 2024b; Chen et al., 2024), web-based task execution (Zhou et al., 2023; Ma et al., 2023), and spreadsheet manipulation (Chen et al., 2025; Ma et al., 2024). Although these systems can perform complex tasks, they remain largely reactive: they initiate actions only in response to explicit user instructions or predefined prompts and do not autonomously interpret environmental signals to determine when action is needed (Kim et al., 2024; Liu et al., 2023). This reactive paradigm limits their applicability in dynamic or open-ended settings where timely intervention is required.

Proactive Assistance Frameworks. Several frameworks explore proactive behavior by allowing systems to anticipate user needs or request clarifications when instructions are ambiguous (Lu et al., 2024; Zhang et al., 2024b). While these approaches represent early steps toward anticipatory assistance, their proactive capabilities are typically limited to information gathering rather than autonomous multi-step task execution. Moreover, proactively determining both *when* to intervene and *what* task to pursue without relying on explicit user commands remains largely unexplored in prior work.

Prompt-based Reasoning Methods. Prompt-based reasoning techniques such as Chain-of-Thought (CoT) (Wei et al., 2022) and self-consistency decoding (Wang et al., 2022) improve problem decomposition and reasoning accuracy. ReAct (Yao et al., 2023b) integrates reasoning with action generation. More expressive structured reasoning methods, including Tree-of-Thoughts (ToT) (Yao et al., 2023a) and Graph-of-Thoughts (GoT) (Besta et al., 2024), explore multiple solution paths but incur substantial computational overhead, making them unsuitable for latency-sensitive LAM settings.

Proactive, Task-Agnostic Reasoning. P-LAM is designed to address these limitations by enabling proactive task initiation directly from environmental observations, without explicit user instructions, fine-tuned demonstrations, or task-specific training. A lightweight intervention classifier determines whether and how to act, and task-type–aligned templates guide execution through a fixed, non-branching reasoning sequence. Unlike CoT-, ToT-, or GoT-based prompting, which introduces variable computational cost through sampling or branching, P-LAM offers consistent latency while enabling proactive behavior. Furthermore, most existing LAM-based systems assume that the task is specified by the user, whereas P-LAM performs implicit task inference, identifying both the need for intervention and the underlying task type. This makes P-LAM a scalable and efficient approach for proactive, task-agnostic reasoning across diverse environments.

3 P-LAM: Proactive Task-Agnostic Reasoning

P-LAM enables proactive decision-making in zero-shot environments by decoupling intervention detection from task-aligned proposal generation. Unlike conventional LAMs that initiate actions only upon explicit user instructions, P-LAM autonomously interprets raw observations, determines whether intervention is required, infers the latent task type, and generates a proactive execution plan through a fixed, non-branching sequence of LLM calls. This section presents the design principles, formal formulation, and complete reasoning pipeline underlying our framework.

3.1 Design Principles

P-LAM is built upon three principles that enforce computational efficiency, task-adaptive reasoning, and robustness to unseen task distributions.

Lightweight Proactive Decision-Making. To support real-time operation, Step 1 must remain lightweight and predictable. It is implemented as a single forward LLM evaluation with no branching, rollouts, or sampling. This yields proactive decisions whose computational cost does not grow with task difficulty or reasoning depth.

Task-Type Conditional Prompting. Effective proactive reasoning requires prompts that align with the underlying structure of the task. P-LAM conditions its proposal generator on a coarse task-type label, $c_t \in \{\text{structured}, \text{unstructured}\}$, which selects between two complementary inductive biases: a compositional, step-wise reasoning style for structured tasks and a more flexible, semantically driven style for unstructured tasks. This design allows the same base model to adapt its reasoning behavior through prompt selection alone, without additional training. The task-type classification is intentionally coarse-grained and is designed to reflect the dominant reasoning structure of a task rather than impose strict boundaries, thereby permitting overlap across domains such as data analysis or mathematics.

Zero-Shot Generalization. P-LAM operates without curated task exemplars or in-context demonstrations. All decisions and proposals must remain stable when observations originate from previously unseen environments:

$$(d_t, c_t, s_t) = g(o_t), \qquad a_t = f(o_t, d_t, c_t, s_t), \qquad o_t \sim \mathcal{D}_{\text{unseen}}.$$

This motivates the design of a task-agnostic prompting interface that generalizes beyond development-time distributions.

3.2 Formalization and Proactive Reasoning Pipeline

We formalize P-LAM's proactive reasoning as a sequential process grounded in intervention detection, task-type inference, and template-guided proposal generation.

3.2.1 Formal Definition

Let $o_t \in \mathcal{O}$ denote the observation at timestep t, where $\mathcal{O}$ includes natural language messages, system logs, or structured event traces. Step 1 predicts whether intervention is necessary, the task type, and a calibrated confidence score:

$$(d_t, c_t, s_t) = g(o_t), \tag{1}$$

where d_t is the intervention decision, $c_t \in \{\text{structured}, \text{unstructured}\}$ is the inferred task type, and $s_t \in [0, 1]$ is the confidence. Intervention is triggered only if $d_t = 1$ and $s_t \geq \tau$ for a predefined threshold τ.

Step 2 generates a task-aligned plan:

$$a_t = f(o_t, d_t, c_t, s_t), \tag{2}$$

where f maps observations and classifier outputs into the action space $\mathcal{A}$. Template selection is governed by a template-mapping function:

$$T : \{\text{structured}, \text{unstructured}\} \to \mathcal{T}, \quad T(c_t) \in \mathcal{T}, \tag{3}$$

where each template $T(c_t)$ is a prompt function instantiated into an LLM call.

The overall policy is defined deterministically as:

$$a_t = \pi(o_t) = f(o_t, d_t, c_t, s_t). \tag{4}$$

3.2.2 Step 1: Proactive Intervention Classifier

As illustrated in Figure 2, P-LAM receives only raw environmental observations rather than explicit user instructions. The Proactive Intervention Classifier must therefore infer latent task-relevant signals and determine whether proactive intervention is warranted.

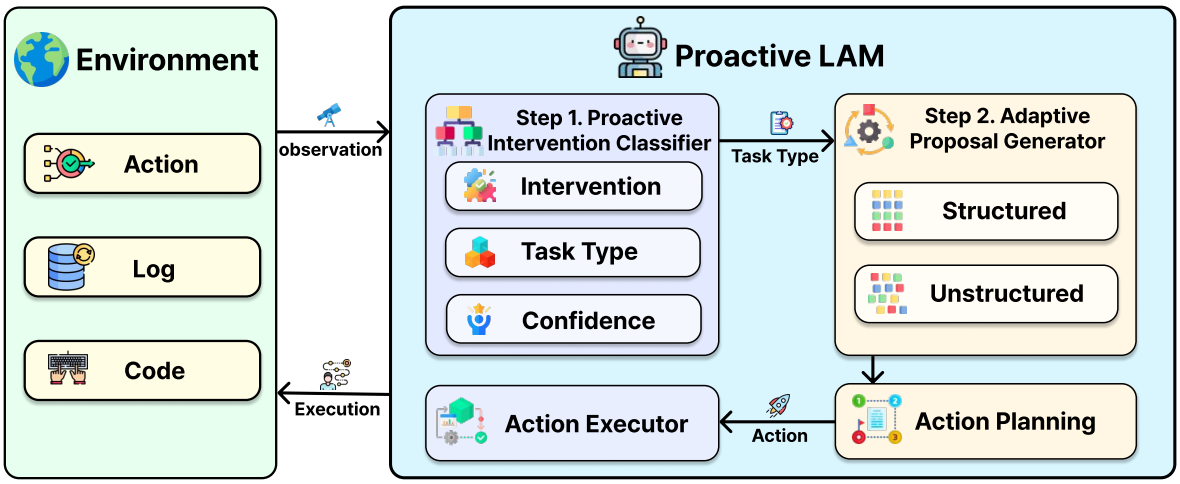

Figure 2: Architecture of P-LAM. The system consists of a *Proactive Intervention Classifier* (Step 1), an *Adaptive Proposal Generator* (Step 2), and an *Action Executor*. Interventions are triggered only when both necessity and confidence criteria are satisfied, after which proposals are generated using a task-type conditional template and executed in the environment.

Context Aggregation. To mitigate noise and partial observability, recent observations are aggregated into a compact situational context:

$$\tilde{o}_t = \mathcal{A}(o_{1:t}), \tag{5}$$

where $\mathcal{A}$ denotes a context aggregation operator realized through a single LLM forward pass.

Latent Situational Representation. From the aggregated context, the classifier derives an internal latent representation encoding inferred goals, progress signals, and potential risk indicators:

$$z_t = \phi(\tilde{o}_t), \tag{6}$$

where ϕ is an implicit reasoning operator implemented within the prompting process rather than a separately trained model.

Intervention Decision. The necessity of proactive intervention is determined by projecting the latent representation onto a decision score:

$$p_t = \sigma(W_d z_t), \qquad d_t = \mathbf{1}\{p_t \geq \epsilon\}, \tag{7}$$

where W_d denotes a conceptual linear projection illustrating how intervention signals are aggregated, rather than a learned parameter, and ϵ is a fixed decision threshold.

Task-Type Inference. In parallel, the classifier infers the task type associated with the current context:

$$c_t = h(o_t) \in \{\text{structured}, \text{unstructured}\}, \tag{8}$$

where h denotes a lightweight task-type predictor implemented via prompting.

Confidence Estimation. A continuous confidence score is estimated from the latent representation:

$$s_t = f_{\text{conf}}(z_t), \tag{9}$$

where f_{conf} produces a scalar reflecting the classifier's reliability in its intervention decision. Among all intermediate signals, s_t is the only explicitly quantified variable and is used to regulate intervention frequency. Step 2 is invoked if and only if $d_t = 1$ and $s_t \geq \tau$, where τ is a predefined confidence threshold. The confidence score s_t itself is directly generated by the LLM as a self-assessed scalar confidence following the explicit instruction in the Step 1 prompt, rather than derived from token probabilities or external calibration.

3.2.3 Step 2: Adaptive Proposal Generator

Given the outputs of Step 1, the Adaptive Proposal Generator produces a task-aligned proactive plan,

$$a_t = f(o_t, d_t, c_t, s_t) = T(c_t)\big(\Phi(o_t)\big), \tag{10}$$

where Φ extracts core reasoning cues from the observation and $T(c_t)$ provides a task-specific inductive bias.

Structured Tasks. For tasks that exhibit compositional logical structure (e.g., program repair, data analysis, or multi-step problem solving), the structured template guides the model through an explicit reasoning scaffold:

$$\begin{aligned} T_{\text{struct}} : \Phi(o_t) \\ \mapsto (\text{analyze}, \text{plan}, \text{synthesize}) \end{aligned} \tag{11}$$

Algorithm 1 P-LAM: Proactive Task-Agnostic Reasoning

Require: Observation o_t, confidence threshold τ
Ensure: Action a_t

Step 1: Proactive Intervention Classifier
$(d_t, c_t, s_t) \leftarrow$ **ProactiveInterventionClassifier**(o_t)
if $d_t = 0$ **or** $s_t < \tau$ **then**
  $a_t \leftarrow \varnothing$ ▷ no proactive action
  return a_t
Step 2: Adaptive Proposal Generator
if $c_t =$ structured **then**
  $T \leftarrow$ **StructuredPrompt**()
else
  $T \leftarrow$ **UnstructuredPrompt**()
$a_t \leftarrow$ **LAM**(T, o_t, c_t)
ActionExecutor(a_t)
return a_t

Decomposition mitigates step-omission errors common in LLM reasoning, and structured synthesis produces an actionable plan that is compatible with downstream tool execution.

Unstructured Tasks. For open-ended tasks such as writing and summarization, the unstructured template emphasizes semantic abstraction rather than rigid decomposition:

$$T_{\text{unstruct}} : \Phi(o_t) \mapsto (\text{interpret, goal, plan, synthesize}) \tag{12}$$

This allows P-LAM to infer high-level objectives, explore alternative formulations, and condense them into a coherent proactive proposal.

Action Execution. The plan is executed via an environment interface that applies code, tool commands, or textual outputs depending on the task domain. This closes the perception–action loop and enables iterative refinement when needed.

3.3 Algorithmic Implementation

Algorithm 1 summarizes the complete P-LAM workflow with integrated confidence scoring.

4 Experiments

We evaluate P-LAM along two complementary dimensions of proactive behavior: (i) **Step 1**: proactive intervention, which determines *when* to intervene and *which* task type to trigger based solely on raw environmental observations; and (ii) **Step 2**: task execution, which evaluates execution quality and zero-shot generalization on contamination-resistant LiveBench after a proactive decision has been made. All experiments are conducted in a strictly zero-shot setting, without any task-specific fine-tuning or additional instruction examples.

4.1 Step 1: Proactive Intervention Evaluation

4.1.1 Task Definition

We follow the ProactiveBench evaluation setup, which is based on the proactive-intervention framework introduced in Proactive Agent (Lu et al., 2024). In the proactive intervention setting, the agent receives a sequence of user observations and must (1) determine whether intervention is necessary, and (2) classify the anticipated task as either *structured* or *unstructured*. Scenarios span coding, writing, and everyday user activities. Dataset statistics and examples are provided in Appendix A.

4.1.2 Dataset and Evaluation Protocol

We follow the official ProactiveBench data split. All reported results use the held-out test set with 233 proactive-intervention events. Confidence thresholds τ are selected exclusively on the validation split and then applied to the test set.

Limitations of Standard F1. Standard F1

$$\text{F1} = \frac{2TP}{2TP + FP + FN}$$

treats false positives and false negatives symmetrically and entirely ignores true negatives. In proactive systems, unnecessary interventions (FP) incur significantly higher user cost than missed interventions. Empirically, over-eager agents can achieve deceptively high F1 by maximizing recall, while safe non-intervention is not rewarded.

Proactive F1 (PF1). To better capture the safety–accuracy trade-off, we define:

$$\text{FAR} = \frac{FP}{FP + TN}, \qquad \text{SafeRate} = 1 - \text{FAR},$$

and compute the harmonic mean of F1 and SafeRate:

$$\text{PF1} = \frac{2 \cdot \text{F1} \cdot \text{SafeRate}}{\text{F1} + \text{SafeRate}}.$$

PF1 penalizes both over-eager (low SafeRate) and overly cautious (low F1) agents.

Model	Setting	Thres.	Precision	Recall	F1	FAR	PF1
Llama-3.1-8B-Instruct	Baseline	–	0.3682	0.9310	0.5277	0.9521	0.0879
	P-LAM	0.90	0.5133	0.6667	0.5800	0.3767	**0.6009**
Qwen-2.5-7B-Instruct	Baseline	–	0.3697	0.8966	0.5235	0.9110	0.1522
	P-LAM	0.90	0.5536	0.7126	0.6231	0.3425	**0.6399**
Llama-3.1-70B-Instruct	Baseline	–	0.3750	0.9310	0.5347	0.9247	0.1321
	P-LAM	0.90	0.6535	0.7586	0.7021	0.2397	**0.7300**
Qwen-2.5-72B-Instruct	Baseline	–	0.3762	0.8736	0.5260	0.8630	0.2174
	P-LAM	0.85	0.6765	0.7931	0.7302	0.2260	**0.7514**
GPT-4.1-mini	Baseline	–	0.3854	0.9080	0.5411	0.8630	0.2186
	P-LAM	0.85	0.6275	0.7356	0.6772	0.2603	**0.7071**

Table 1: Baseline vs confidence-optimized intervention results. Threshold values correspond to the PF1-maximizing confidence setting for each model. FAR: False Alarm Rate. SafeRate = $1-$FAR.

4.1.3 Intervention Results

Applying confidence thresholding to Step 1 predictions consistently reduces false alarms while preserving or slightly improving standard F1. Across all evaluated model families, threshold optimization yields substantial gains in PF1, indicating that calibrated intervention confidence is crucial for safe proactive agents.

A full breakdown of ΔF1, ΔPF1, ΔFAR, and relative PF1 gains for each model is provided in Appendix B.2.

4.1.4 Threshold Behavior

Despite producing similar F1 scores across a wide range of thresholds, all models exhibit pronounced PF1 sensitivity to confidence thresholding. Across architectures, we observe two robust patterns B.3

- **F1 is nearly invariant** to thresholding, confirming its inadequacy as a confidence-based evaluation metric for proactive systems.
- **PF1 peaks sharply** for $\tau \in [0.8, 0.9]$, indicating the optimal safety–accuracy operating region.

Larger models tend to display larger PF1 gains when applying confidence thresholding, suggesting more stable confidence estimates that P-LAM can effectively exploit during intervention decisions.

4.1.5 Task-Type Routing Accuracy

To assess the quality of Step 1 task-type prediction, we evaluate structured vs. unstructured classification accuracy on the ProactiveBench test split. Table 2 summarizes routing accuracy across model families. Overall, large models (Llama-3.1-70B, Qwen-2.5-72B) achieve over 90% accuracy, while smaller models exhibit mild degradation. A detailed confusion-matrix analysis is provided in Appendix C.

Model	Overall Acc.	Struct. Acc.	Unstruct. Acc.
Llama-3.1-70B-Instruct	93.98%	99.05%	89.19%
Qwen-2.5-72B-Instruct	90.55%	97.98%	83.33%
GPT-4.1-mini	79.90%	98.06%	62.26%
Llama-3.1-8B-Instruct	79.72%	84.76%	75.00%
Qwen-2.5-7B-Instruct	78.29%	75.00%	82.67%

Table 2: Task-type routing accuracy on ProactiveBench.

4.2 Step 2: Zero-Shot Generalization on LiveBench

We now evaluate Step 2 of P-LAM, which governs the quality of task execution after an intervention is triggered. This experiment examines the model's ability to generalize to unseen tasks across the six LiveBench domains. Whereas Step 1 determines *when* and *how* to route control, Step 2 evaluates *how effectively* the selected template solves the resulting task.

4.2.1 Main Results

Table 3 summarizes the performance of P-LAM across all model families. P-LAM consistently improves performance in all domains, particularly in reasoning, mathematics, data analysis, and coding—tasks requiring explicit multi-step structure.

A striking outcome is that P-LAM enables the smaller Qwen2.5-7B-Instruct to achieve a LiveBench score of 61.31%, surpassing the score of the much larger Qwen2.5-72B-Instruct under

Model	LiveBench Score	Reasoning	Math	Language	Instruction Following	Data Analysis	Coding
Llama-3.1-8B-Instruct	28.50	17.30	14.10	35.00	44.15	38.25	22.50
+Few-shot (k=3)	45.20	22.30	38.50	55.40	61.80	68.20	27.90
+SC (k=8)	40.30	25.80	26.60	42.00	60.80	55.80	30.90
+P-LAM	49.38	26.80	46.20	53.40	57.40	63.00	49.50
Qwen2.5-7B-Instruct	35.00	33.00	34.63	23.43	32.83	46.23	39.90
+Few-shot (k=3)	52.70	38.00	51.18	48.02	50.55	83.17	45.31
+SC (k=8)	47.50	42.00	45.30	35.60	56.00	53.40	52.70
+P-LAM	61.31	39.31	62.60	67.20	55.40	87.20	56.20
Llama-3.1-70B-Instruct	45.20	45.30	28.00	45.30	67.40	54.90	34.60
+Few-shot (k=3)	58.40	50.30	52.40	65.70	79.95	79.80	39.70
+SC (k=8)	56.20	53.80	43.00	52.30	78.90	69.80	45.60
+P-LAM	66.15	58.50	60.70	57.30	71.20	91.60	57.60
Qwen2.5-72B-Instruct	50.30	47.00	37.40	34.90	37.40	54.90	56.60
+Few-shot (k=3)	68.00	52.00	61.80	67.30	63.00	79.70	62.00
+SC (k=8)	62.80	56.00	52.40	54.90	68.40	70.80	67.60
+P-LAM	75.20	62.20	63.90	87.20	66.10	97.30	74.60
GPT-4.1-mini	59.05	53.78	58.78	38.00	70.31	61.34	72.11
+Few-shot (k=3)	71.80	58.78	73.16	70.40	83.86	86.11	77.52
+SC (k=8)	70.50	63.78	70.28	55.00	81.81	75.34	83.61
+P-LAM	79.30	94.70	63.80	71.80	73.00	97.90	75.10

Table 3: LiveBench results for open-source models and GPT-4.1-mini. LiveBench Score denotes the average over all evaluation domains. All models use CoT as the baseline. Few-shot (k=3), SC (CoT-SC, k=8), and P-LAM are applied as additional prompting strategies.

CoT prompting 50.30%. This demonstrates that adaptive prompting can significantly narrow (or even reverse) model-scale performance gaps.

Figure 3 further analyzes the impact of template selection.

4.2.2 Template Ablation: Structured vs. Unstructured vs. Adaptive

To isolate the effect of task-type adaptivity, we compare structured-only prompting, unstructured-only prompting, and P-LAM's classifier-driven adaptive prompting. Figure 3 reports per-domain accuracy.

The results show that the structured template performs best in mathematics, reasoning, and coding tasks, while the unstructured template yields higher accuracy in the remaining language-oriented domains. P-LAM's adaptive prompting combines these complementary strengths, resulting in more consistent performance across all evaluation categories.

Structured prompting is most effective for tasks requiring explicit multi-step decomposition, whereas unstructured prompting is advantageous for semantic and open-ended tasks. P-LAM's adaptive strategy combines these complementary strengths.

4.3 Performance–Token Trade-off

While the primary evaluation in Section 4.2 focuses on accuracy across LiveBench domains, we additionally examine the token usage associated with each prompting strategy. Since inference cost scales approximately linearly with token count, tokens provide a practical and model-agnostic proxy for execution cost. Rather than reporting full cost curves for all model families, we present a compact analysis centered on Qwen2.5-7B, the most representative mid-sized model in our evaluation suite.

Table 4 summarizes the relationship between accuracy improvements and total token consumption for four prompting strategies. Few-shot prompting achieves a sizable accuracy gain (+17.70), but the additional in-context examples inflate the token count by 3.90× compared to the baseline. Self-consistency further increases token usage to 8.00× due to multi-sampling, despite offering only a moderate performance improvement (+12.50). In contrast, P-LAM yields the largest accuracy increase (+26.31) while maintaining a near-baseline token footprint (1.31×), indicating a substantially more favorable accuracy–token balance.

Overall, this analysis highlights that P-LAM not

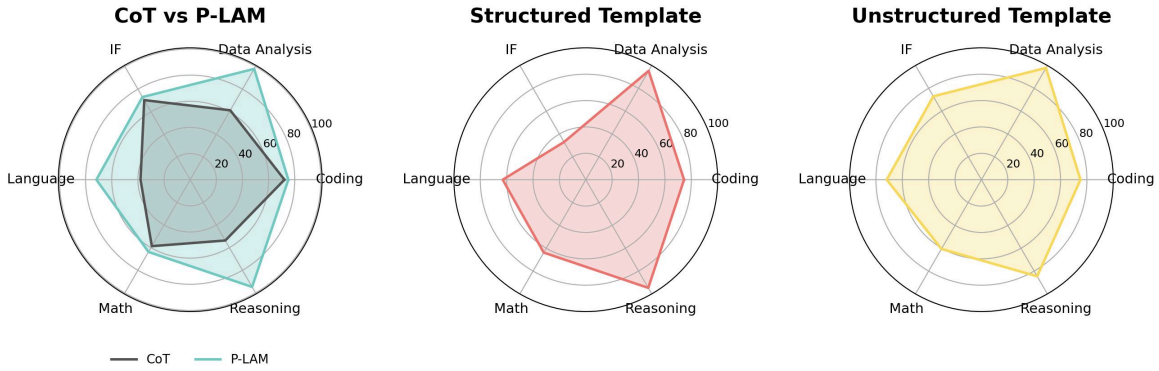

Figure 3: Ablation of structured-only, unstructured-only, and adaptive prompting. Structured templates excel in analytical domains (reasoning, math, coding), while unstructured templates perform better in open-ended language tasks. Adaptive prompting achieves the most consistent performance across all domains.

Strategy	Acc.	Gain	Tokens	Mult.
CoT (Baseline	35.00	–	986.8	1.00×
Few-shot (k=3)	52.70	+17.70	3,847.4	3.90×
SC (k=8)	47.50	+12.50	7,893.6	8.00×
P-LAM	**61.31**	**+26.31**	**1,288.0**	**1.31×**

Table 4: Accuracy–token trade-off for Qwen2.5-7B.

only improves task performance but does so with a token budget that remains close to the baseline. This property makes P-LAM particularly attractive in settings where inference efficiency, latency, or cost are practical constraints a scenario common in real-world LAM deployments.

5 Conclusion

We presented P-LAM, a proactive, task-agnostic reasoning framework that enables language models to determine *when* to intervene and *how* to act using a two-step prompting design. Our approach introduces a confidence-aware intervention mechanism and a task-type–adaptive proposal generator, requiring no fine-tuning or task-specific examples.

In Step 1, confidence calibration substantially improved proactive reliability, yielding consistently higher PF1 scores while reducing false alarms across all evaluated models. In Step 2, P-LAM improved zero-shot task execution on LiveBench across reasoning, mathematics, data analysis, and coding settings. Notably, P-LAM allowed a 7B model to match or exceed the performance of a 72B baseline under standard CoT prompting, suggesting that adaptive prompting can partially offset differences in model scale.

Overall, P-LAM demonstrates that proactive decision-making and task-type adaptivity are key ingredients for more reliable and versatile action models. We hope this work supports future research on proactive systems that are both more capable and more aligned with user intent.

Limitations

While P-LAM substantially improves proactive reliability, several limitations remain. Because proactive systems may intervene without explicit user requests, inappropriate or premature actions can pose safety and usability risks. In practical deployments, additional safeguards such as conservative intervention thresholds or human-in-the-loop verification may therefore be necessary.

Our evaluation focuses on text-based interaction environments, and extending proactive reasoning to multimodal or tool-rich settings may require additional robustness mechanisms. Although this minimal structure proved effective across evaluated domains, richer interaction settings may benefit from further architectural refinement.

Finally, Step 1 relies on LLM-internal confidence signals, which are treated as lightweight heuristics rather than calibrated uncertainty estimates. Such confidence estimates may vary across deployment settings or degrade under distribution shift, potentially leading to premature interventions or missed opportunities to act. Similarly, while PF1 captures important safety–accuracy trade-offs, it does not reflect long-term user preferences or downstream effects of proactive interventions. Future work should explore richer uncertainty modeling, user-in-the-loop feedback, and broader proactive benchmarks.

Ethical Considerations

P-LAM introduces proactive intervention mechanisms that enable language models to initiate actions without explicit user instructions. While such proactive behavior can improve efficiency and user experience, it also raises ethical and societal considerations related to autonomy, safety, and potential misuse.

A primary risk of proactive systems is over-intervention. If confidence estimates are miscalibrated or environmental signals are ambiguous, the model may interrupt users unnecessarily, reduce user autonomy, or propose inappropriate actions. This risk is particularly salient in safety-critical or high-stakes settings, where premature or incorrect interventions could lead to negative outcomes. To mitigate this risk, P-LAM explicitly incorporates confidence-aware intervention thresholding, which significantly reduces false alarms in our evaluation. Nevertheless, reliable calibration remains an open challenge under distribution shift.

Another consideration concerns differential impact across user populations. Users with non-standard interaction patterns, limited technical expertise, or accessibility needs may be disproportionately affected by proactive assistance, potentially leading to exclusion or unfair treatment. Our evaluation is limited to text-based benchmarks and does not capture the full diversity of real-world user behavior. Future work should incorporate user-centered evaluations and feedback mechanisms to better understand these effects.

In practical deployments, it is also important to ensure transparency and user control, such that users are aware of proactive interventions and can override or opt out of them when desired.

We emphasize that P-LAM is intended as a research prototype. The framework is evaluated in controlled benchmark environments and is not designed for direct deployment without human oversight. Responsible use of proactive action models should include appropriate safeguards, monitoring mechanisms, and clear boundaries on permissible actions, especially when operating in open-ended or real-world environments.

Despite these risks, we believe proactive reasoning is an important research direction, as many real-world systems require timely assistance in the absence of explicit user instructions, and understanding its limitations is a prerequisite for responsible deployment.

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pages 17682–17690.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Yibin Chen, Yifu Yuan, Zeyu Zhang, Yan Zheng, Jinyi Liu, Fei Ni, Jianye Hao, Hangyu Mao, and Fuzheng Zhang. 2025. Sheetagent: towards a generalist agent for spreadsheet reasoning and manipulation via large language models. In *Proceedings of the ACM on Web Conference 2025*, pages 158–177.

Yuxiao Chen, Jingzheng Wu, Xiang Ling, Changjiang Li, Zhiqing Rui, Tianyue Luo, and Yanjun Wu. 2024. When large language models confront repository-level automatic program repair: How well they done? In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, pages 459–471.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. 2022. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35:18343–18362.

Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. 2024. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models. *arXiv preprint arXiv:2403.07714*.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, and 1 others. 2024. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, and 1 others. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

Yaxi Lu, Shenzhi Yang, Cheng Qian, Guirong Chen, Qinyu Luo, Yesai Wu, Huadong Wang, Xin Cong, Zhong Zhang, Yankai Lin, and 1 others. 2024. Proactive agent: Shifting llm agents from reactive responses to active assistance. *arXiv preprint arXiv:2410.12361*.

Kaixin Ma, Hongming Zhang, Hongwei Wang, Xiaoman Pan, Wenhao Yu, and Dong Yu. 2023. Laser: Llm agent with state-space exploration for web navigation. *arXiv preprint arXiv:2309.08172*.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. 2024. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37:94871–94908.

Theo X Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2023. Is self-repair a silver bullet for code generation? *arXiv preprint arXiv:2306.09896*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. 2024. Gorilla: Large language model connected with massive apis. *Advances in Neural Information Processing Systems*, 37:126544–126565.

Gabriele Piccoli, Joaquin Rodriguez, and Anas Mahmoud. 2025. Large action models for programmatic orchestration. *Business & Information Systems Engineering*, pages 1–12.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*.

Lu Wang, Fangkai Yang, Chaoyun Zhang, Junting Lu, Jiaxu Qian, Shilin He, Pu Zhao, Bo Qiao, Ray Huang, Si Qin, and 1 others. 2024a. Large action models: From inception to implementation. *arXiv preprint arXiv:2412.10047*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, and 1 others. 2024. Livebench: A challenging, contamination-limited llm benchmark. *arXiv preprint arXiv:2406.19314*.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

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

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Quoc Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, and 1 others. 2025. xlam: A family of large action models to empower ai agent systems. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11583–11597.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. 2024a. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. *arXiv preprint arXiv:2401.07339*.

Xuan Zhang, Yang Deng, Zifeng Ren, See-Kiong Ng, and Tat-Seng Chua. 2024b. Ask-before-plan: Proactive language agents for real-world planning. *arXiv preprint arXiv:2406.12639*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A Prompting Setup for P-LAM

This appendix summarizes the prompt templates used by P-LAM and briefly explains the design choices behind the two-step prompting scheme.

A.1 Two-Step Prompting Overview

P-LAM separates proactive assistance into:

- **Step 1 (Proactive Intervention)**: Given recent observations o_t, decide whether to intervene, predict a coarse task type (structured vs. unstructured), and output a confidence score.
- **Step 2 (Task Execution)**: Conditioned on the task type, select either a structured or unstructured template to solve the downstream task.

This separation (i) aligns Step 1 with proactive-agent benchmarks (intervention decision quality), (ii) allows Step 2 to focus purely on task performance (e.g., LiveBench), and (iii) exposes an explicit confidence score that can be thresholded to trade off coverage vs. unnecessary interventions.

A.2 Step 1 Prompt: Proactive Intervention Classifier

Design Rationale. The Step 1 prompt decomposes the decision into six stages:

1. **Summarize Observations**: aggregate recent actions into a compact summary.
2. **Infer User's Goal**: make the user's immediate goal explicit.
3. **Assess User's State**: categorize progress (e.g., progressing vs. struggling).
4. **Intervention Decision**: decide whether proactive help is appropriate *now*.
5. **Task-Type Classification**: route to a structured vs. unstructured template in Step 2.
6. **Confidence Assessment**: output a scalar confidence for calibration and thresholding.

This structure encourages the model to (i) separate "what the user is doing" from "how well it is going," and (ii) produce simple, interpretable outputs (`should_intervene`, `task_type`, `confidence_score`) that can be plugged into downstream policies.

A.3 Step 2 Prompts: Domain-Adaptive Task Execution

Design Rationale. Step 2 uses two simple templates:

- **Structured template**: for coding, data-analysis, and math tasks that benefit from explicit algorithmic planning (*My Analysis* $\rightarrow$ *My Plan* $\rightarrow$ *Final Solution*).
- **Unstructured template**: for language-heavy tasks (reasoning, instruction following, language comprehension) that require careful instruction parsing and content transformation.

The task-type label from Step 1 (`"structured"` vs. `"unstructured"`) is used only to choose between these two templates; the underlying base model remains unchanged. This keeps the routing simple and interpretable while still allowing domain-appropriate reasoning structure.

A.3.1 Structured Template

A.3.2 Unstructured Template

B Benchmark Details and Confidence Analysis

B.1 Benchmark Summary

Table 8 summarizes the benchmarks used in our experiments and their correspondence to each step of P-LAM.

Evaluation Decoupling. Together, ProactiveBench and LiveBench decouple the evaluation of *when to act* (Step 1: proactive intervention decision) from *how to act* (Step 2: downstream task execution). This separation allows us to isolate proactive decision quality from task-solving performance, enabling a more precise analysis of each component in the P-LAM framework.

B.2 PF1 Improvements from Confidence Thresholding

We report detailed PF1 improvements obtained by applying confidence thresholding to the Step 1 proactive intervention classifier. For each model, we compare the baseline (un-thresholded) setting against the PF1-maximizing confidence threshold selected on the validation split. We report the change in standard F1 (ΔF1), PF1 (ΔPF1), false alarm rate (ΔFAR), as well as the relative PF1 gain.

As shown in Table 9, confidence thresholding consistently yields substantial PF1 improvements across all model families. These gains are primarily driven by large reductions in the false alarm rate, while standard F1 is preserved or slightly improved in all cases. This indicates that safety-oriented calibration effectively suppresses unnecessary interventions without sacrificing overall intervention accuracy.

Notably, the large relative PF1 gains reflect the fact that un-thresholded models tend to over-intervene, leading to high false alarm rates. Confidence thresholding enables P-LAM to operate in a more conservative and reliable regime, which is critical for proactive systems where unnecessary interventions incur high user cost.

B.3 Confidence Calibration

We treat the confidence score from Step 1 as a calibrated probability and apply a global threshold

Step 1 Prompt: Proactive Intervention Classifier

You are a world-class proactive AI assistant. Your goal is to deeply understand the user's context and intent in order to determine **if** and **how** you should intervene.

Process:

1. **Summarize Observations**: Briefly summarize the user's recent actions.
2. **Infer User's Goal**: Based on the summary, identify the user's most likely immediate goal.
3. **Assess User's State**: Determine whether the user is progressing, exploring, struggling, or comparing options. Repetitive actions, context switching, or new searches are strong indicators that assistance may be needed.
4. **Intervention Decision & Rationale**: Decide whether intervention is appropriate and provide clear justification.
5. **Task-Type Classification**: If intervening: "structured" or "unstructured". If not intervening: `null`.
6. **Confidence Assessment (0.0–1.0)**: 0.5–0.6 = low, 0.7–0.8 = moderate, 0.85–0.9 = high, 0.95–1.0 = very high. If not intervening, return 0.0.

Output a JSON object:

```
{
  "reasoning": "...chain of thought...",
  "should_intervene": true/false,
  "task_type": "structured" | "unstructured" | null,
  "confidence_score": float
}
```

Table 5: Full prompt for Step 1 proactive intervention classification.

τ to decide whether to intervene. For each model, we sweep τ and report PF1, F1, precision, recall, and SafeRate.

C Task-Type Routing Analysis

Figure 5 shows the confusion matrices for task-type classification across all model families. While large models (Llama-3.1-70B, Qwen-2.5-72B) achieve balanced performance across both structured and unstructured categories, smaller models occasionally exhibit noticeable asymmetries.

In particular, GPT-4.1-mini demonstrates a strong bias toward predicting structured tasks: its accuracy for structured cases reaches 98.06%, but drops to 62.26% for unstructured cases. We quantify this imbalance via a simple bias metric

$$\text{Bias} = |\text{Acc}_{\text{struct}} - \text{Acc}_{\text{unstruct}}|.$$

GPT-4.1-mini displays the largest bias (0.3579), exceeding the next highest model by more than a factor of three. This suggests that routing errors for smaller models stem primarily from under-recognition of unstructured scenarios, which can mislead the Step 2 prompt selection.

Importantly, this bias does not typically result in catastrophic task failure. Instead, it limits the effectiveness of adaptive prompting by routing unstructured tasks to structured templates, thereby reducing potential gains from task-type alignment. Addressing this routing asymmetry remains an important direction for improving proactive systems, particularly for smaller models.

Structured Template (for Coding & Reasoning & Math)

You are a world-class expert problem solver. Begin by carefully analyzing the task, then organize your reasoning using "My Analysis" and "My Plan" before presenting the final answer.

Task Input (JSON): {json_problem}

1. **My Analysis**: Restate the task objective, key inputs, desired outputs, and relevant constraints. Include any reasoning required to interpret the problem clearly.
2. **My Plan & Core Method**: Outline the detailed step-by-step solution strategy or algorithm. Describe the core principle or method that justifies each step in the plan.
3. **Final Solution**: Provide only the final answer. Explanations or reasoning should not appear here.

Table 6: Structured template used for coding, data-analysis, and math tasks.

Unstructured Template (for Language, Data Analysis, Instruction Following)

Carefully analyze the problem and provide a structured solution.

Task Input (JSON): {json_problem}

1. **Instruction Analysis**: Rephrase the problem, extract essential information, and identify any implicit constraints.
2. **Ultimate Objective**: Describe the final goal the user seeks to achieve.
3. **Principle Extraction & Execution Plan**: Identify the relevant principles or rules and outline a clear multi-step plan.
4. **Execution & Final Answer**: Follow the plan and provide only the final answer in the required format.

Table 7: Unstructured template used for reasoning, language, and math tasks.

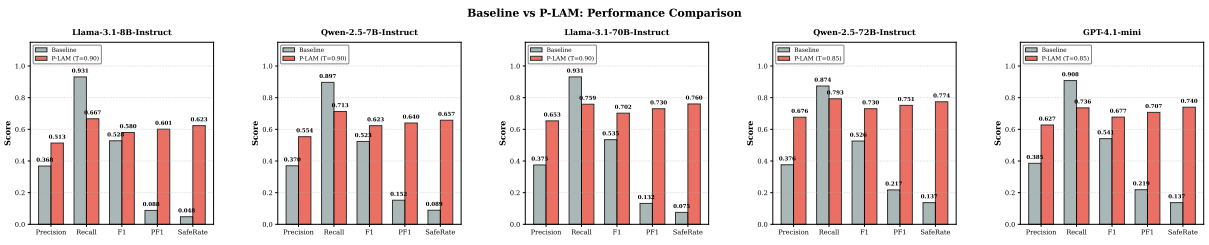

Figure 4: Confidence threshold analysis across all evaluated models. PF1, F1, precision, recall, and SafeRate are plotted as functions of the threshold. The PF1-optimal point is marked with a black star.

Benchmark Summary: ProactiveBench and LiveBench

ProactiveBench
Step Used: Step 1 (Intervention Decision)
Instances: 6,790 events (train), 233 events (test)
Categories: 4 domains
Description: ProactiveBench is an event-driven proactive-intervention benchmark designed to evaluate an agent's ability to determine *when* a user requires assistance based on contextual cues and temporal activity patterns. It consists of a large-scale training set of 6,790 event sequences and a real-world test set of 233 human-collected event traces. Each event sequence reflects realistic, temporally grounded computer-use behavior through a mixture of simulated environments and real activity references.
The data span four domains—coding (2,275 events), writing (2,354 events), daily-life computer interaction (2,161 events), and general task workflows—capturing a broad spectrum of user behaviors. The test set comprises 233 real-world events across 12 fine-grained scenarios and serves as an authentic evaluation setting for assessing whether an agent can accurately detect genuine user need while avoiding unnecessary or premature interventions.
Overall, ProactiveBench provides rich temporal grounding and diverse contextual structures, enabling rigorous assessment of proactive decision-making quality.

LiveBench (Version: 2024-11-30)
Step Used: Step 2 (Task Execution)
Instances: Not fixed (monthly updated)
Categories: 6 categories
Description: LiveBench is a continually updated evaluation suite covering six major categories: math, coding, reasoning, language, instruction-following, and data analysis. Because tasks are refreshed monthly, the exact number of instances varies per release; we use the 2024-11-30 version in this work. LiveBench is specifically designed to minimize contamination and to provide fresh, diverse queries for evaluating task execution capability under real-world conditions.

Table 8: Benchmark summary for ProactiveBench and LiveBench used in our experiments.

Model	ΔF1	ΔPF1	ΔFAR	**Relative PF1 Gain**
Llama-3.1-8B-Instruct	+0.0523	+0.5130	-0.5754	+584.0%
Qwen-2.5-7B-Instruct	+0.0996	+0.4877	-0.5685	+320.4%
Llama-3.1-70B-Instruct	+0.1674	+0.5979	-0.6850	+452.5%
Qwen-2.5-72B-Instruct	+0.2042	+0.5340	-0.6370	+245.6%
GPT-4.1-mini	+0.1361	+0.4885	-0.6027	+223.5%

Table 9: PF1 improvements from confidence thresholding. Relative PF1 Gain measures the proportional improvement in PF1 after applying the PF1-maximizing confidence threshold for each model.

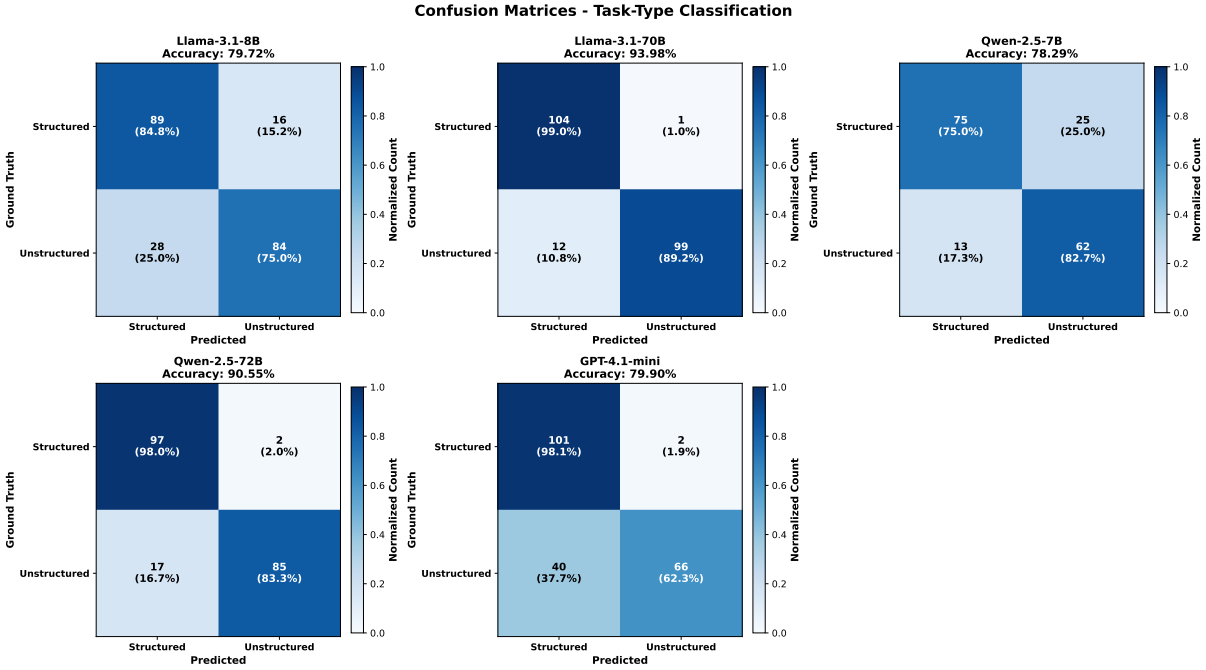

Figure 5: Confusion matrices for task-type routing across all models.