TIDE: Proactive Multi-Problem Discovery via Template-Guided Iteration

Anonymous ACL submission

Abstract

Agents are widely deployed as assistants over documents, tools, and code. However, they typically act only on explicit user requests, which surface only the problems the user has noticed, while many other important problems coexist, hidden in plain sight, within the broader user context, with their total number unknown in advance. We frame this as the task of *discovering multiple hidden problems from context*, in which coexisting problems should be uncovered, grounded in supporting evidence, and paired with concrete actions. To this end, we introduce TIDE, a template-guided iterative framework with two complementary mechanisms. Specifically, motivated by the observation that single-pass prediction anchors on the most salient cases and yields generic claims, we propose *iterative discovery*, which surfaces a small batch of candidates per round while conditioning on what has already been found, so subsequent rounds extend coverage; and *thought templates*, reusable schemas distilled from previously solved cases that specify what contextual signals to attend to and how to connect them, anchoring each prediction in a recognizable problem class. We validate TIDE on two realistic settings, personal workspaces and software repositories, across four model backbones, showing substantial gains over single-shot and parallel multi-agent baselines on task coverage, identification, and resolution.

1 Introduction

Large language model (LLM) agents are now routinely deployed as digital assistants that read documents, invoke external tools, and operate over complex environments (Yao et al., 2023; Yang et al., 2024a; Wu et al., 2024; Baek et al., 2025). Yet despite their growing capability, these agents remain reactive: they act only after a user issues an explicit request, whether to summarize a file, schedule a meeting, fix a failing test, or invoke a particular tool. This interaction model presumes that the user already knows what is wrong and what to ask.

In practice, however, the most consequential issues are often the ones a user has not yet noticed: a budget approval given verbally but not yet recorded in writing, holding up a vendor order on a hard deadline; two copies of the same report with conflicting numbers, both heading into an upcoming review; a recurring meeting the team has tacitly stopped attending, still blocking the only window for an urgent kickoff. Such issues sit, often in plain sight, inside the very documents, emails, and calendar entries that the agent could in principle inspect.

These otherwise different issues share a common structure that extends beyond the workspace setting above. Across the digital environments where agents operate (a personal workspace, a software repository, or another rich working context), evidence accumulates in which multiple problems coexist; none is articulated as a request; the number of coexisting problems is not known in advance; and resolving only the most salient one leaves the rest untouched. We therefore argue that proactive assistance is best framed not as anticipating a single user intent, but as the broader task of *discovering multiple hidden problems from context*. Existing work on proactive agents has studied when to intervene (Liu et al., 2025; Zhang et al., 2025) or how to anticipate a single localized need (Lu et al., 2025; Yang et al., 2025a; Pasternak et al., 2025), but largely sidesteps this multi-problem, context-wide setting that real workflows demand.

Meeting this task requires two complementary capabilities: broad coverage over coexisting problems that compete for attention with more salient ones, and enough precision per candidate to be actionable rather than speculative. To address these challenges, we present TIDE (**T**emplate-guided **I**terative **D**iscovery and r**E**solution; Figure 1), a framework that combines two mechanisms operating along distinct axes. Specifically, motivated

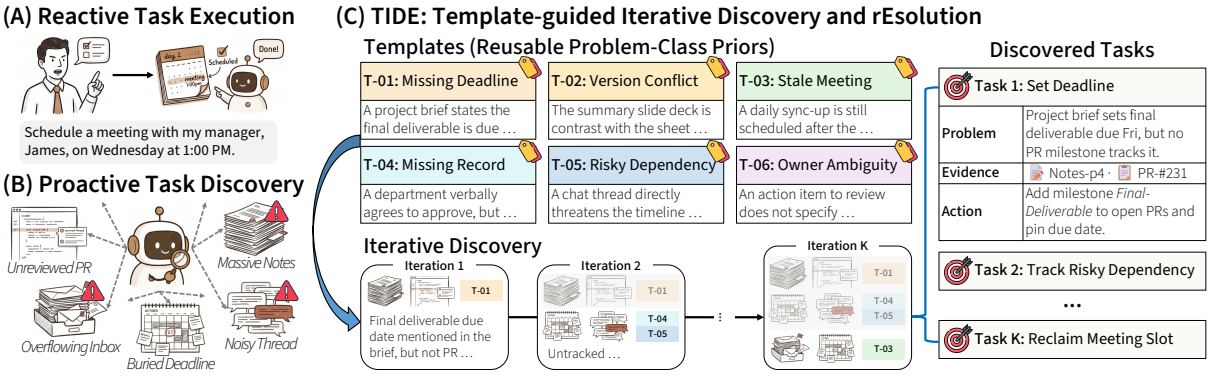

Figure 1: Conceptual illustration of TIDE. (A) Reactive agents act only on explicit user requests, leaving (B) the many problems coexisting hidden across the user context untouched. (C) TIDE surfaces them by applying reusable thought templates over multiple rounds of iterative discovery conditioned on the cumulative state, returning per-task plans that identify, ground, and resolve each discovered problem.

by the observation that single-pass prediction anchors on the most salient cases and yields generic claims, we propose *iterative discovery*: rather than producing all predictions in one pass, the agent surfaces a small batch of candidates per round while conditioning on what has already been found, so that subsequent rounds are pushed beyond the cumulative discovery state. Additionally, for every surfaced candidate, the agent retrieves the supporting artifacts that serve as evidence and commits to a concrete action that proposes a resolution, so that the per-case output forms a multi-element plan that simultaneously identifies, grounds, and addresses each of the discovered problems. We then complement iteration with *thought templates*: reusable schemas distilled from previously solved cases, each capturing a recognizable pattern of hidden problem and laying out the chain of contextual signals from which that pattern can be inferred, anchoring each prediction in a known problem class instead of leaving the agent to infer it from scratch. Iteration thus expands the set of problems the agent will consider, while templates provide a reusable prior on how problems manifest in evidence, sharpening each prediction.

We instantiate this TIDE framework in two different real-world settings that share the underlying multi-problem structure: personal workspaces, where the agent surfaces multiple unresolved bottlenecks from workspace documents and proposes how to address each, and software repositories, where the agent identifies multiple hidden bugs directly from source code and produces patches that resolve them. Across both settings and four LLM backbones, TIDE consistently outperforms single-shot and parallel multi-agent baselines on task coverage, identification, and resolution; analyses further show that iterative discovery and thought templates contribute largely complementary gains and that templates transfer across backbones. Taken together, these results suggest that proactive assistance is better cast as an explicit, multi-step discovery process over context than as single-shot prediction from a user request, and they offer a general recipe for building agents that can both surface what users would not have thought to ask and point toward how to address it.

2 Method

Our method is motivated by three observations: (1) the hidden problems within a given context are typically multiple in unknown number, and the most salient ones systematically overshadow subtler ones, so single-shot prediction leaves most problems undiscovered; (2) generic prompting offers no reusable prior on how contextual signals become evidence for a particular class of problem, leading predictions to drift toward generic or speculative claims; and (3) such patterns recur across instances and can be distilled from previously solved cases for reuse. Guided by these observations, we first formalize the task (Section 2.1) and then describe our framework, TIDE (**T**emplate-guided **I**terative **D**iscovery and r**E**solution), which couples iterative discovery for broader coverage with thought templates for sharper fidelity (Section 2.2).

2.1 Preliminaries: Hidden-Problem Discovery from Context

Task Formulation We consider an agent operating over a collection of documents $\mathcal{D}$, broadly the digital artifacts available to the agent (e.g., emails

and documents in a personal workspace, or functions in source code from a software repository). Within $\mathcal{D}$ there exists a latent set of *hidden problems*, formalized as follows:

$$\mathcal{P}^\star = \{p_1^\star, p_2^\star, \ldots, p_n^\star\},$$

where none of which is articulated as an explicit user request and whose cardinality n is not known in advance. The objective is to produce a predicted set $\hat{\mathcal{P}}$ that approximates $\mathcal{P}^\star$, where each prediction takes the form of a triple $\hat{p} = (b, \hat{\mathcal{D}}, a)$ comprising a natural-language description b of the candidate problem, a supporting subset $\hat{\mathcal{D}} \subseteq \mathcal{D}$ that grounds the prediction in evidence, and a concrete action a that proposes a resolution. Solution quality therefore turns on two complementary axes: the *coverage* over hidden problems in $\mathcal{P}^\star$, and the per-prediction *fidelity*, with b correctly describing the problem, $\hat{\mathcal{D}}$ providing valid grounding, and a proposing an effective resolution.

Single-Shot Discovery Given this task, a natural baseline is to prompt a large language model to produce all problems in a single pass:

$$\hat{\mathcal{P}} = \texttt{LLM}(\mathcal{D}).$$

Yet, this formulation has two failure modes that correspond to the quality axes above: salient problems overshadow subtler ones, capping coverage; without any prior on what kinds of evidence patterns indicate a problem, predictions tend to drift toward generic or speculative claims, eroding fidelity.

2.2 Template-Guided Iterative Discovery

To address these failure modes of single-shot discovery, we propose TIDE that couples two complementary components: *thought templates*, reusable schemas distilled from previously solved cases that sharpen per-prediction fidelity, and *iterative discovery*, which applies these templates over multiple rounds while conditioning on the cumulative state, broadening coverage over coexisting problems.

Thought Templates Rather than having the agent infer evidence patterns from scratch, we distill such patterns from prior cases into reusable discovery schemas, which we call thought templates. Specifically, each template specifies (i) a *name* that labels a recurring class of hidden problem, (ii) a *pattern* stating the structural form of that class, and (iii) an *evidence flow*, an ordered sequence of contextual signals to attend to and how they should be connected to infer instances of that class.

Formally, let $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$ denote the template set, where each template is a tuple:

$$t_i = (\text{name}_i,\ \text{pattern}_i,\ \text{evidence flow}_i). \quad (1)$$

Templates are constructed once from a collection of solved cases (in training instances) and held fixed at inference. To be more specific, for each solved case $\langle \mathcal{D}_{\texttt{train}}, p_{\texttt{train}}, r_{\texttt{train}} \rangle$ comprising the relevant document collection, the previously discovered problem, and a reference resolution (e.g., a patch in the software-repository setting), we prompt an LLM to abstract away instance-specific details and emit a template in the structured form of Equation 1: $t_i = \texttt{LLM}(\mathcal{D}_{\texttt{train}}, p_{\texttt{train}}, r_{\texttt{train}})$. At inference, the full set $\mathcal{T}$ is supplied to the agent as a library of discovery schemas, so that each prediction can be anchored in a recognizable problem class rather than inferred from scratch. An illustrative template from the workspace setting is below.

Workspace Template: Conflicting Source-of-Truth Blocks Sign-off Under Deadline

Pattern: A shared source artifact exists in conflicting versions across channels, and an imminent deadline blocks sign-off until one is made authoritative.

Evidence flow: (i) locate the deliverable and its cited source; (ii) find conflicting copies across channels and confirm a material discrepancy; (iii) tie the conflict to a time-bounded review and the owner who can resolve it.

Iterative Discovery and Resolution However, it is worth noting that even when equipped with such a template library, prompting the agent to produce all discoveries at once still concentrates its capacity on the most salient cases, leaving the subtler ones uncovered. To address this, we instead let the agent surface predictions over multiple rounds, with each round explicitly conditioned on what has already been found, so that subsequent rounds are pushed beyond the cumulative discovery state.

Formally, let $\hat{\mathcal{P}}^{(t)}$ denote the cumulative prediction state after round t, initialized as $\hat{\mathcal{P}}^{(0)} = \emptyset$. In round t, the agent generates a small batch of up to k new candidate predictions in the triple form $(b, \hat{\mathcal{D}}, a)$ defined in Section 2.1, conditioned on the document collection $\mathcal{D}$, the template set $\mathcal{T}$, and the previous state $\hat{\mathcal{P}}^{(t-1)}$, formalized as follows:

$$\Delta\hat{\mathcal{P}}^{(t)} = \texttt{LLM}\Big(\mathcal{D}, \mathcal{T}, \hat{\mathcal{P}}^{(t-1)}, k\Big). \quad (2)$$

The state is then updated as $\hat{\mathcal{P}}^{(t)} = \hat{\mathcal{P}}^{(t-1)} \cup \Delta\hat{\mathcal{P}}^{(t)}$, and the loop terminates after T rounds or earlier if a round returns empty, with the final output returned as $\hat{\mathcal{P}} = \hat{\mathcal{P}}^{(T)}$. We note that, by coupling identification with retrieval and a proposed action inside each per-prediction step, every round emits an actionable plan that simultaneously identifies, grounds, and addresses each surfaced problem, rather than treating these as separate downstream stages.

3 Experimental Setup

We evaluate TIDE on the multi-problem discovery task formalized in Section 2.1, whose goal is to surface multiple coexisting problems from a context $\mathcal{D}$ alone. Below, we describe our datasets, methods, evaluation metrics, and implementation details.

3.1 Datasets

We consider two real-world settings that share the underlying multi-problem structure, a personal workspace and a software repository; for each, we construct an evaluation split by extending existing data sources, since no existing benchmark directly targets multi-problem discovery from context.

Personal Workspace In this setting, each instance is the digital workspace of an individual user, consisting of a profile that captures the role, working style, current priorities, pain points, and relationships of that user, together with the workspace artifacts (documents, emails, and calendar entries) that constitute the context $\mathcal{D}$. Each problem is typically grounded in multiple workspace artifacts, so identifying it requires the agent to piece together evidence across several documents, emails, and calendar entries rather than reading it off any single artifact. The remaining artifacts in $\mathcal{D}$ act as distractors that look plausibly related to ongoing projects, relationships, and work, but are not implicated in any actual problem. A resolution takes the form of an action drawn from a predefined action set (such as sending an email, scheduling a meeting, sharing a document, or escalating to a manager), together with the parameters required to execute it (e.g., the recipients, subject, and body of an outgoing email). To instantiate this setting at scale, we adopt the data construction pipeline of Pasternak et al. (2025) and produce 150 problems across 30 multi-problem workspaces, with 4–6 problems and 88–113 candidate artifacts per workspace.

Software Repository In this setting, each instance is a snapshot of a real-world open-source software repository at a commit where multiple unresolved bugs coexist and fixing them requires producing patches to multiple functions across the codebase. Each problem corresponds to an issue filed by a real GitHub user against the repository, and its gold resolution is the code patch from the pull request that fixed the issue. The context $\mathcal{D}$ is the set of candidate functions parsed from the snapshot; only a subset of these contains the coexisting bugs, while the rest are distractor functions that appear at the same snapshot but are not implicated in any bug. To instantiate this setting, we collect GitHub issues from Python repositories in SWE-BENCH (Jimenez et al., 2024) and TESTEXPLORA (Liu et al., 2026), and group same-repository issues at a common anchor commit at which the buggy functions of every grouped issue are unfixed; keeping only groups with at least two coexisting bugs spanning at least two buggy functions yields 146 problems across 20 multi-bug test instances drawn from 11 projects, with 2–41 problems and 6–646 candidate functions per instance.

3.2 Methods

We compare the following methods, all of which use the same backbone LLM (supporting the long-context) and operate over the same context $\mathcal{D}$, with the full $\mathcal{D}$ placed directly in the context window:

- **SINGLE-AGENT:** Generates multiple problem predictions in a single LLM pass over $\mathcal{D}$.
- **MULTI-AGENT:** Runs multiple independent LLM agents in parallel over $\mathcal{D}$, matched in number to the rounds used by our iterative discovery.
- **TIDE** (Ours): Combines iterative discovery with thought templates, conditioning each round on the cumulative discovery state.

3.3 Evaluation Metrics

Since each instance contains multiple gold problems and the model surfaces multiple predictions, our metrics score individual gold-prediction pairs and aggregate them into instance-level scores.

Scoring Components Each matched (gold, prediction) pair is scored along three components: *retrieval*, *identification*, and *resolution*. Specifically, retrieval is measured by the overlap between the predicted and gold-annotated evidence IDs, while identification and resolution are scored by an LLM judge (Liu et al., 2023) on a Likert-style rubric

	Methods	Workspace						Repository						Avg.
		Retrieval		Identification		Resolution		Retrieval		Identification		Resolution		
		Cov.	F1	Cov.	F1	Cov.	F1	Cov.	F1	Cov.	F1	Cov.	F1	
GPT	SINGLE-AGENT	47.60	54.32	47.85	54.63	49.67	56.14	8.66	10.34	11.15	12.92	12.19	13.27	31.56
	MULTI-AGENT	32.15	45.41	27.24	38.85	29.64	41.85	10.11	12.66	10.19	12.77	9.89	12.39	23.59
	TIDE (Ours)	**69.06**	**70.46**	**67.64**	**68.76**	**76.08**	**77.32**	**16.82**	**18.61**	**17.29**	**19.73**	**15.52**	**17.39**	**44.56**
Gemini	SINGLE-AGENT	50.01	61.48	42.99	52.95	35.95	44.31	13.08	17.20	13.34	16.90	15.08	18.71	31.83
	MULTI-AGENT	46.10	58.54	38.98	49.95	32.44	41.54	14.75	18.51	14.71	17.72	15.32	18.89	30.62
	TIDE (Ours)	**83.84**	**84.91**	**70.11**	**71.05**	**54.37**	**55.08**	**22.55**	**25.14**	**21.93**	**24.22**	**24.32**	**26.98**	**47.04**
Claude	SINGLE-AGENT	13.82	30.73	13.53	25.74	17.69	22.00	12.85	16.09	9.86	12.18	9.84	12.34	16.39
	MULTI-AGENT	21.60	43.33	18.04	36.04	23.99	31.04	9.37	13.36	7.96	11.18	8.88	11.01	19.65
	TIDE (Ours)	**32.01**	**55.51**	**35.77**	**62.44**	**46.49**	**54.88**	**19.99**	**22.70**	**14.50**	**16.50**	**15.79**	**17.76**	**32.86**
Qwen	SINGLE-AGENT	30.46	42.05	32.44	44.67	28.60	37.60	5.60	6.83	5.34	6.62	4.84	5.76	20.90
	MULTI-AGENT	39.34	52.12	31.04	42.27	26.21	35.37	5.00	6.72	6.58	7.50	5.83	6.50	22.04
	TIDE (Ours)	**52.39**	**60.21**	**50.50**	**58.13**	**41.87**	**48.06**	**9.94**	**11.33**	**6.87**	**8.07**	**8.47**	**9.70**	**30.46**

Table 1: Main results on the two evaluation settings: Personal Workspace and Software Repository. For each sub-task, we report Coverage (Cov.) and F1 over three independent runs; the best per-LLM results are in **bold**.

against the gold problem description and the gold reference action, respectively (See Appendix A).

Coverage and F1 To aggregate per-pair scores into instance-level metrics, we use retrieval as the matching score and pair each gold problem with its best-scoring prediction, as well as each prediction with its best-scoring gold. The coverage of a component is then the matched score averaged over all gold problems, capturing how well the agent discovers each of the hidden problems. In addition to this, the F1 of a component is the harmonic mean of this coverage and the analogous matched score averaged over all predictions. Notably, when multiple predictions match the same gold, only the one with the highest retrieval score is credited when averaging over predictions, while the rest contribute zero, penalizing extraneous predictions. Both metrics are macro-averaged across instances.

3.4 Implementation Details

We instantiate the agent with each of four LLMs that support long-context: GPT-5 mini (OpenAI, 2025), Claude Sonnet 4.5 (Anthropic, 2025), Gemini 3.5 Flash (Google DeepMind, 2026), and Qwen 3.6 Flash (Qwen Team, 2026). The LLM judge is fixed to GPT-5 mini. Thought templates are constructed by each LLM on its own from a held-out set of solved training cases (disjoint from the test split), yielding 40 templates for the personal-workspace setting and 108 templates for the software-repository setting. For iterative discovery, we set $T = 10$ rounds for the personal-workspace setting and $T = 3$ for the software-repository setting; in both cases, we condition each round on the cumulative state $\hat{\mathcal{P}}^{(t-1)}$ and terminate early when a round returns an empty batch.

4 Results and Analyses

4.1 Main Results

Table 1 shows performance on retrieval, identification, and resolution across the Workspace and Repository settings, under four LLM backbones. Although recent LLMs can attend over the entire candidate pool in a single long-context pass, this access alone is insufficient for multi-problem discovery. This is reflected in the SINGLE-AGENT baseline, which recovers only the most salient items and leaves the majority of gold problems undiscovered. Specifically, without revisiting the context, the agent commits to its first hypotheses and rarely expands beyond them, exposing a limitation of single-shot prediction. Likewise, running the same backbone as multiple independent agents in parallel does not close this gap, as independent runs tend to converge on the same items rather than complement each other. In contrast, TIDE combines iterative discovery with reusable templates, consistently achieving the best performance.

4.2 In-Depth Analyses

Discovery on Multi-Problem Instances. Our framework is designed for multi-problem instances,

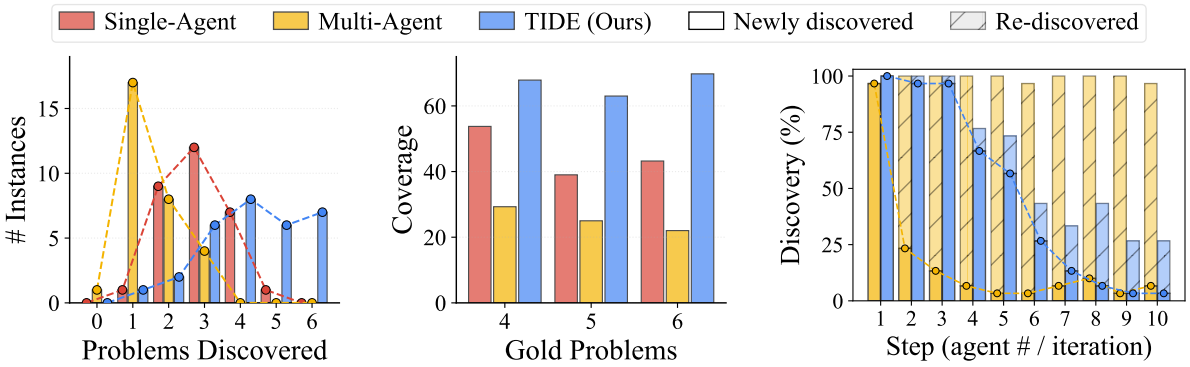

Figure 2: Multi-problem discovery on the Workspace setting with GPT. Left: discovered problems per instance. Right: coverage by gold count.

Figure 3: Newly vs. re-discovered predictions on the Workspace task with GPT.

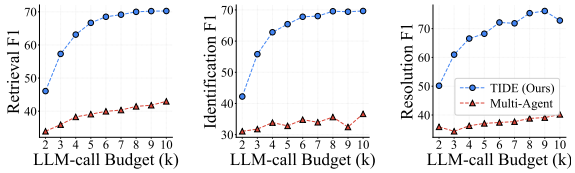

Figure 4: F1 results as a function of the per-instance LLM-call budget k on the Workspace setting with GPT.

Method	Retrieval		Identification		Resolution	
	Cov.	F1	Cov.	F1	Cov.	F1
SINGLE-AGENT	8.66	10.34	11.15	12.92	12.19	13.27
ITER. + DEMOS	10.40	11.43	11.09	12.80	12.71	12.80
TIDE (Ours)	**16.82**	**18.61**	**17.29**	**19.73**	**15.52**	**17.39**

Table 2: Results on the Repository setting with GPT using raw few-shot demonstrations (ITER. + DEMOS).

where recovering a single problem is not enough. To directly assess this capability, we report results in Figure 2, where every instance contains four to six gold problems. Specifically, both baselines mostly discover only one or two problems per instance (left), while TIDE often reaches four or more. Moreover, across instances with increasing numbers of gold problems (right), TIDE continues to recover most of them while the baselines lag further behind, with MULTI-AGENT falling below the simpler SINGLE-AGENT in this setting. Notably, MULTI-AGENT underperforms SINGLE-AGENT despite its larger budget, and we examine this next.

Effectiveness of Iterative Discovery. To understand why MULTI-AGENT fails to deliver comparable gains under the same LLM-call budget as TIDE, we decompose each step's prediction into two categories: newly discovered and re-discovered. As shown in Figure 3, both methods start from the same point at the first step, but from the second step onward, MULTI-AGENT exhibits a significant drop in newly discovered items and remains low across subsequent steps, while re-discovered items take over the majority of its predictions. TIDE, by contrast, keeps contributing newly discovered problems across the following iterations, and the re-discovered share grows only gradually as the gold pool saturates. Each agent in MULTI-AGENT runs without access to what others have surfaced, so it re-anchors on the same most-salient signal and re-issues it. In contrast, conditioning each round on the cumulative discovery state redirects capacity toward additional problems that independent parallel agents leave uncovered, driving the coverage gap in Table 1.

Effect of LLM-Call Budget. The diversity analysis explains why MULTI-AGENT stops accumulating new problems, but it leaves open whether a larger budget could close the gap. To address this, we vary the per-instance LLM-call budget k from 2 to 10, where k corresponds to the iteration cutoff for TIDE and to the number of aggregated parallel agents for MULTI-AGENT. As shown in Figure 4, TIDE scales steeply with k, while MULTI-AGENT plateaus early. Interestingly, MULTI-AGENT at k=10 still falls below TIDE at k=2. This suggests that the gains of iterative discovery extend beyond retrieval coverage to identification and resolution, and more fundamentally, that scaling parallel agents is no substitute for iterative conditioning.

Effectiveness of Thought Templates. Having shown that iterative discovery enables TIDE to keep surfacing new problems across rounds, we now turn to what templates contribute. To this end, we ablate templates from TIDE and track retrieval coverage and precision at each iteration cutoff. Figure 6 shows that the template-guided variant yields a small additional coverage gain over the

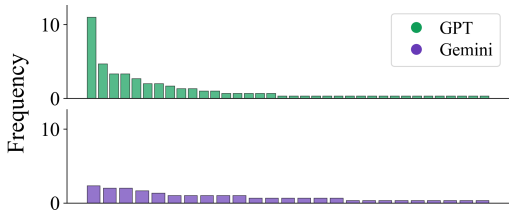

Figure 5: Per-run template citation frequency.

Inference	Templates	Retrieval Cov.	Retrieval F1	Identification Cov.	Identification F1	Resolution Cov.	Resolution F1
GPT	GPT	16.82	12.12	17.29	11.81	15.52	11.31
	Gemini	16.30	11.60	15.31	10.03	18.36	12.38
Gemini	Gemini	21.47	17.69	20.63	17.05	23.22	19.23
	GPT	24.03	19.03	22.84	17.86	24.70	19.19

Table 3: Template transferability on the Repository setting.

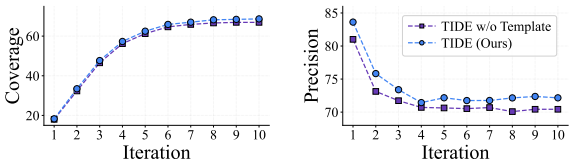

Figure 6: Per-iteration retrieval coverage (left) and precision (right) on the Workspace setting with GPT.

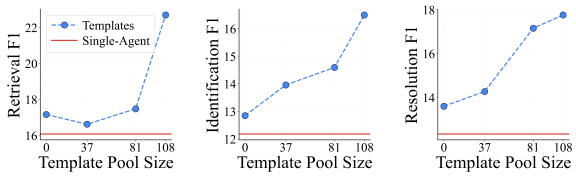

Figure 7: F1 scores as the template pool size grows on the repository setting with Claude.

no-template ablation (left) and a more pronounced precision margin at every iteration (right). Taken together with the diversity analysis, this shows that iteration and templates play complementary roles: iteration mainly drives how much of the gold pool is recovered, while templates mainly drive how accurate each recovered item is.

Few-Shot as Template Substitute. We next ask whether few-shot demonstrations can replace the thought templates in TIDE. To examine this, we follow the same iterative setup as TIDE but replace its thought templates with raw few-shot demonstrations drawn from the same training pool used to construct the templates. As shown in Table 2, demonstrations within the iterative loop fall well short of TIDE across retrieval, identification, and resolution. This indicates that the value of templates lies in abstracting past examples into reusable reasoning patterns rather than in merely exposing the agent to them.

Template Usage Distribution Across LLMs. We next examine how each backbone draws from its own template pool during inference. Figure 5 sorts cited templates by per-run usage frequency for each LLM, where bar height is averaged across runs. The two backbones differ in concentration. GPT concentrates more sharply on a few recurring templates, whereas Gemini spreads citations more evenly across its cited pool. These divergent usage patterns raise a natural question: do templates built by one backbone still transfer to another?

Cross-LLM Template Transferability. To address this, we fix the inference LLM and vary the template source between GPT and Gemini, as shown in Table 3. Across all three components, transferred templates perform comparably to self templates in both directions, indicating that templates remain reusable across backbones despite each LLM's distinct usage profile.

Effect of Template Pool Size. To examine how performance scales with the size of the template pool, we vary the number of available templates and report results in Figure 7. Notably, iteration alone already outperforms SINGLE-AGENT on all three axes, and adding templates yields further gains that grow with the pool size.

4.3 Qualitative Study

Beyond the quantitative gains shown above, we further provide a qualitative analysis on two representative cases, one from each evaluation setting. In the workspace case (Table 4), the gold issue is that the volunteer-tracking platform double-counts Mar 8 *Community Build Day* check-ins, and the vendor patch is blocked behind a pending IT Security access approval ahead of a Mar 20 senior-leadership briefing. SINGLE-AGENT surfaces only an unrelated facility-rider procurement stall and retrieves none of the gold documents, so the identification, the chosen action, and the addressee are all wrong. TIDE, by contrast, reaches the data-integrity issue in a later iteration, retrieves the gold documents, and escalates to the right manager with the gating access ticket, the vendor-deployment deadline, and the presentation deadline. In the repository case (Table 5), the gold issue is a multi-function bug in *mlxtend*'s `mlxtend/evaluate/mcnemar.py`: the paired helpers `mcnemar_table` and `mcnemar_tables`

both populate the 2×2 McNemar contingency table with mirrored off-diagonal assignments, so the fix has to swap `tb[1, 0]` and `tb[0, 1]` in step across both constructors. SINGLE-AGENT returns the two paired helpers as two isolated single-function bottlenecks, fixing each in place but never naming the shared pattern that ties them together. TIDE, guided by a mirrored-index-assignment template, retrieves both gold constructors inside a single bottleneck and frames the swap as one coupled defect to repair in step, recovering the multi-function fix site that the single-pass agent splits apart. In both settings, TIDE's prediction is guided by a thought template that captures a pattern recurring across instances within the same domain.

5 Related Work

Task-oriented LLM Agents LLM agents have been increasingly studied in task-oriented environments that require document understanding (Ma et al., 2024), tool use (Schick et al., 2023; Qin et al., 2024), web interaction (Zhou et al., 2024; Deng et al., 2023), or software engineering (Yang et al., 2024a; Zhang et al., 2024b). A growing body of benchmarks and systems (Liu et al., 2024; Xie et al., 2024) evaluates whether agents can follow user instructions, navigate complex environments, and complete prescribed tasks. These settings, however, typically presume that the task has already been specified through a user request, issue description, failing test, or otherwise localized goal, reducing the role of the agent to executing against a stated objective. In contrast, we target the inverse setting in which no such request exists and the relevant problems, often multiple and coexisting, first need to be discovered from a broader context before any of them can be acted on.

Proactive Agents Proactive agents aim to move beyond the reactive interaction model by anticipating user needs and initiating assistance before an explicit request is issued. One line of work focuses on uncovering user intent that goes beyond what is literally stated, either by asking clarification questions to resolve ambiguous requests (Aliannejadi et al., 2019; Kuhn et al., 2022; Zhang et al., 2024a; Sun et al., 2025) or by navigating knowledge gaps that have not been articulated (Kaur et al., 2026); these approaches, however, still presume a user-issued query as the anchor of interaction. A more recent line broadens the scope, studying when an agent should intervene (Liu et al., 2025; Zhang et al., 2025), how user activity or signals can be used to anticipate assistance opportunities (Lu et al., 2025; Yang et al., 2025a), and how proactive suggestions should be generated and surfaced (Pasternak et al., 2025). Yet across this literature, proactivity remains anchored to a single localized need at a time, leaving open how an agent should jointly surface, ground, and resolve the many coexisting problems that real workflows typically contain.

Templates for LLM Reasoning Improving LLM reasoning has traditionally relied on internal model capability, whether by eliciting intermediate steps (Wei et al., 2022; Kojima et al., 2022) or by having a model critique and revise its own outputs (Madaan et al., 2023; Shinn et al., 2023). A more recent work observes that useful reasoning patterns recur across problems and externalizes them as reusable templates that can be retrieved and applied: Buffer-of-Thoughts (Yang et al., 2024b) caches prior reasoning traces for retrieval on new problems, and follow-up work extends this idea to hierarchical template paths (Yang et al., 2025b,c) and to multi-hop reasoning over long-context documents (Jeong et al., 2026). These template-based approaches, however, share a common assumption that the problem statement is already given and templates serve as schemas for how to solve it. We instead repurpose templates as discovery schemas that specify what contextual signals to attend to and how to connect them in order to infer a problem that has not been stated, and apply them iteratively so that each round extends coverage over coexisting problems rather than refines a single solution.

6 Conclusion

We presented TIDE, a framework for discovering multiple hidden problems from context through iterative discovery and thought templates. Across personal workspaces and software repositories, TIDE consistently outperforms single-shot and multi-agent baselines on retrieval, identification, and resolution, with iteration and templates contributing complementary gains and templates transferring across backbones. In particular, iteration drives coverage by redirecting capacity toward undiscovered problems, while templates sharpen each prediction by anchoring it in a recognizable problem class. We believe these findings recast proactive assistance as a multi-step discovery process over context, offering a recipe for agents that surface what users would not have thought to ask.

Limitations

Our TIDE delivers consistent gains across two realistic settings and four backbones, and the design choices behind it open a couple of interesting directions for further work. First, templates are built once from a pool of solved cases and remain fixed at inference, which already proves effective and transfers across backbones; however, updating the library online from agent interactions, or augmenting the pool with automatically constructed cases, are natural extensions. Likewise, iterative discovery trades a small bounded budget for broader coverage, a trade-off our analyses show is favorable against multi-agent baselines at matched budgets, and further investigating this iterative paradigm would be an exciting direction for future work.

Ethics Statement

Our TIDE is designed to assist users by surfacing hidden problems from their working context that would otherwise go unaddressed, ranging from overlooked bottlenecks in personal workspaces to coexisting bugs in software repositories. Since the framework operates over real-world documents and distills templates from previously solved cases, both of which may carry sensitive, biased, or otherwise undesirable content depending on the underlying source, we recommend applying standard safeguards such as content filtering, bias detection, and human-in-the-loop review at both template construction and deployment, in line with best practices for responsibly deploying LLM-based agents.

References

Mohammad Aliannejadi, Hamed Zamani, Fabio Crestani, and W. Bruce Croft. 2019. Asking clarifying questions in open-domain information-seeking conversations. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, pages 475–484. ACM.

Anthropic. 2025. Claude Sonnet 4.5 system card. Technical report, Anthropic.

Jinheon Baek, Sujay Kumar Jauhar, Silviu Cucerzan, and Sung Ju Hwang. 2025. Researchagent: Iterative research idea generation over scientific literature with large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, pages 6709–6738. Association for Computational Linguistics.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samual Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Google DeepMind. 2026. Gemini 3.5 Flash model card. Technical report, Google DeepMind.

Soyeong Jeong, Taehee Jung, Sung Ju Hwang, Joo-Kyung Kim, and Dongyeop Kang. 2026. When thoughts meet facts: Reusable reasoning for long-context lms. *Findings of ACL 2026*, abs/2510.07499.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Kirandeep Kaur, Vinayak Gupta, Aditya Gupta, and Chirag Shah. 2026. Proper agents: Proactivity driven personalized agents for advancing knowledge gap navigation.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Lorenz Kuhn, Yarin Gal, and Sebastian Farquhar. 2022. Clam: Selective clarification for ambiguous questions with generative language models.

Steven Liu, Jane Luo, Xin Zhang, Aofan Liu, Hao Liu, Jie Wu, Ziyang Huang, Yangyu Huang, Yu Kang, and Scarlett Li. 2026. Testexplora: Benchmarking llms for proactive bug discovery via repository-level test generation. *arXiv preprint arXiv:2602.10471*, abs/2602.10471.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, and 3 others. 2024. Agentbench: Evaluating llms as agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Xingyu Bruce Liu, Shitao Fang, Weiyan Shi, Chien-Sheng Wu, Takeo Igarashi, and Xiang 'Anthony' Chen. 2025. Proactive conversational agents with inner thoughts. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems,*

CHI 2025, YokohamaJapan, 26 April 2025- 1 May 2025, pages 184:1–184:19. ACM.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-eval: NLG evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 2511–2522. Association for Computational Linguistics.

Yaxi Lu, Shenzhi Yang, Cheng Qian, Guirong Chen, Qinyu Luo, Yesai Wu, Huadong Wang, Xin Cong, Zhong Zhang, Yankai Lin, Weiwen Liu, Yasheng Wang, Zhiyuan Liu, Fangming Liu, and Maosong Sun. 2025. Proactive agent: Shifting LLM agents from reactive responses to active assistance. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Yubo Ma, Yuhang Zang, Liangyu Chen, Meiqi Chen, Yizhu Jiao, Xinze Li, Xinyuan Lu, Ziyu Liu, Yan Ma, Xiaoyi Dong, Pan Zhang, Liangming Pan, Yu-Gang Jiang, Jiaqi Wang, Yixin Cao, and Aixin Sun. 2024. MMLONGBENCH-DOC: benchmarking long-context document understanding with visualizations. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

OpenAI. 2025. GPT-5 system card.

Gil Pasternak, Dheeraj Rajagopal, Julia White, Dhruv Atreja, Matthew Thomas, George Hurn-Maloney, and Ash Lewis. 2025. Beyond reactivity: Measuring proactive problem solving in LLM agents. *arXiv preprint arXiv:2510.19771*, abs/2510.19771.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Qwen Team. 2026. Qwen3.6.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Weiwei Sun, Xuhui Zhou, Weihua Du, Xingyao Wang, Sean Welleck, Graham Neubig, Maarten Sap, and Yiming Yang. 2025. Training proactive and personalized LLM agents. *arXiv preprint arXiv:2511.02208*, abs/2511.02208.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2024. Autogen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Bufang Yang, Lilin Xu, Liekang Zeng, Kaiwei Liu, Siyang Jiang, Wenrui Lu, Hongkai Chen, Xiaofan Jiang, Guoliang Xing, and Zhenyu Yan. 2025a. Contextagent: Context-aware proactive LLM agents with open-world sensory perceptions. *Advances in Neural Information Processing Systems*, abs/2505.14668.

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

38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024.

Ling Yang, Zhaochen Yu, Bin Cui, and Mengdi Wang. 2025b. Reasonflux: Hierarchical LLM reasoning via scaling thought templates. *arXiv preprint arXiv:2502.06772*, abs/2502.06772.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. 2024b. Buffer of thoughts: Thought-augmented reasoning with large language models. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Minkai Xu, Joseph E. Gonzalez, Bin Cui, and Shuicheng Yan. 2025c. Supercorrect: Advancing small LLM reasoning with thought template distillation and self-correction. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Xuan Zhang, Yang Deng, Zifeng Ren, See-Kiong Ng, and Tat-Seng Chua. 2024a. Ask-before-plan: Proactive language agents for real-world planning. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, Findings of ACL, pages 10836–10863. Association for Computational Linguistics.

Yichi Zhang, Xin Luna Dong, Zhaojiang Lin, Andrea Madotto, Anuj Kumar, Babak Damavandi, Joyce Chai, and Seungwhan Moon. 2025. Proactive assistant dialogue generation from streaming egocentric videos. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pages 12044–12068. Association for Computational Linguistics.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024b. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024*, pages 1592–1604. ACM.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

A Prompts

We list the four prompts used end-to-end in our pipeline: template construction (Figures 8 and 9) and per-iteration inference (Figures 10 and 11), each instantiated separately for the workspace and code settings. Placeholders in `{curly braces}` are filled at runtime; the *previously found bottlenecks* block is omitted at iteration 1 and re-injected on subsequent iterations.

	Retrieval	Identification	Resolution
Workspace. A corporate community-impact manager's workspace; the volunteer-tracking platform double-counts Mar 8 *Community Build Day* check-ins, and the vendor's fix is blocked behind a pending IT Security access approval ahead of the Mar 20 senior leadership briefing.			
Gold	5 supporting documents: the Mar 12 vendor support ticket from Emily Zhang, the Mar 13 AccessHub access-review request AR-2025-4411, the duplicated VolunteerHub records for the Mar 8 *Community Build Day*, the Mar 20 senior-leadership pre-read deck, and the coordination thread on the data freeze.	VolunteerHub double-counts Mar 8 check-ins because of a sync conflict between SSO and the manual check-in override, so reported volunteer hours are inflated; the vendor patch is staged but cannot deploy until IT Security signs off on AR-2025-4411, and uncorrected figures would land in the Mar 20 senior-leadership briefing.	`escalate_to_manager` to David Chen, summarizing the duplication, the gating AccessHub ticket, the Mar 17 data freeze, and the Mar 20 readout so he can unblock IT Security in time for the vendor patch.
SINGLE-AGENT	× **0 of 5** gold documents (returns only a procurement-intake stall on the Apr 10 Showcase facility rider and a calendar double-booking, neither touching the VolunteerHub problem).	× Surfaces an unrelated facility-rider procurement intake that has been sitting unassigned since Feb 28; the VolunteerHub data-integrity issue is not surfaced at all.	× `send_email` to the procurement intake about the facility rider; the action does not target David Chen, the AccessHub ticket, or the Mar 20 metrics deadline.
TIDE (Ours)	✓ **5 of 5** gold documents, no spurious documents. (iteration 3)	✓ Names the VolunteerHub Mar 8 duplication, ties it to the AccessHub ticket AR-2025-4411 stalled in Security, and links the chain to the Mar 20 senior-leadership briefing.	✓ `escalate_to_manager` to David Chen with a sharp summary of the inflation, the gating Security approval, the Mar 18 vendor deployment window, and the Mar 20 readout.
TIDE template used: `[TID_11] Unblocking high-stakes deliverable under SME communication lag.` *Pattern:* A time-bounded executive deliverable depends on a corrected artifact whose fix is staged but gated behind a slower internal approval or SME response; escalating up the manager chain unblocks the gating step in time for the deliverable.			

Table 4: Case study from the personal-workspace setting (a corporate community-impact manager whose volunteer-tracking platform double-counts *Community Build Day* check-ins ahead of a senior-leadership briefing). Rows are the three methods (Gold, SINGLE-AGENT, TIDE) and columns are the three sub-tasks (retrieval, identification, resolution). ✓ marks a sub-task the method handles correctly; × marks a wrong or missing one. The shaded row at the bottom shows the thought template that TIDE retrieved for this prediction.

	Retrieval	Identification	Resolution
Repository. *mlxtend* issue #393, a multi-function bug in `mlxtend/evaluate/mcnemar.py`: the paired helpers `mcnemar_table` and `mcnemar_tables` both populate the 2×2 contingency table with swapped off-diagonal assignments, so a fix has to land in both constructors consistently.			
Gold	2 gold buggy functions in `mlxtend/evaluate/mcnemar.py`: `mcnemar_table` and `mcnemar_tables`, both of which build the McNemar contingency table.	Each constructor computes `minus_true = m1_vs_true - m2_vs_true` and then assigns `tb[1, 0] = sum(minus_true == 1)` and `tb[0, 1] = sum(minus_true == -1)`; per the docstrings, `tb[0, 1]` should hold the model-1-correct/model-2-wrong count, so the off-diagonals are mirrored in both builders and must be swapped in step.	Patch `mcnemar_table` and `mcnemar_tables` together so that `tb[0, 1]` receives `sum(minus_true == 1)` and `tb[1, 0]` receives `sum(minus_true == -1)` in both constructors.
SINGLE-AGENT	× Retrieves the paired constructors as **two separate bottlenecks**, one containing only `mcnemar_table` and the other only `mcnemar_tables`, treating them as unrelated bugs at isolated sites.	× Each isolated prediction names the swap correctly inside its own function, but the shared cross-function pattern, that the same mirrored assignment lives in two paired helpers and must be fixed in step, is not surfaced.	× Emits two independent patches as if the bugs were unrelated; the edits happen to be locally correct but the coupling is not represented as a single fix.
TIDE (Ours)	✓ Retrieves **both** `mcnemar_table` and `mcnemar_tables` inside a single bottleneck, recovering the full multi-function fix site.	✓ Identifies the shared swap in both constructors, ties `tb[1, 0]` and `tb[0, 1]` back to the documented McNemar layout, and frames the two functions as a coupled fix rather than two coincidences.	✓ Emits one coordinated patch that swaps the off-diagonal assignments in both `mcnemar_table` and `mcnemar_tables` so the contingency table matches the documented layout.
TIDE template used: `[TID_47] Mirrored Index Assignment Across Paired Constructors.` *Pattern:* Two sibling factory functions build the same fixed-shape container (matrix cell, dict slot, record field) for related inputs; both compute a discriminator and assign it to the same index positions, but the index ordering on the assignment is inverted relative to the documented layout. The failure is silent at the call site and only surfaces in a downstream consumer that reads the container under the documented contract.			

Table 5: Case study from the software-repository setting (*mlxtend*, a multi-function bug in the McNemar contingency-table constructors). Rows are the three methods (Gold, SINGLE-AGENT, TIDE) and columns are the three sub-tasks (retrieval, identification, resolution). ✓ marks a sub-task the method handles correctly; × marks a wrong or missing one. The shaded row at the bottom shows the thought template that TIDE retrieved for this prediction.

Template Construction Prompt (Workspace).
You are an expert at extracting reusable reasoning patterns from solved examples.
Given a solved bottleneck detection example, extract an abstract reasoning template.

Solved example shown to the model:

```
Bottleneck: {bottleneck_description}

Evidence Documents Used:
- [{doc.type}]
  {doc.payload as JSON}
...

Checklist Steps:
- Retrieval:       {retrieval_description}
- Identification:  {identification_description}
- Task Execution:  {task_description}
```

Rules:

1. **Domain-agnostic**: use general workplace language only; replace specific domain terms with role-based descriptions.
2. Be **concise**.
3. **Aggressive abstraction**: the template must apply to many scenarios beyond the solved example.
   - Replace specific artifacts (e.g., "allocation spreadsheet", "quarterly slide deck") with generic types (e.g., "a shared source artifact", "a deliverable").
   - Replace specific people, teams, or titles (e.g., "analytics lead", "VP of Marketing") with abstract role descriptors (e.g., "the content owner", "the approver").
   - Replace specific events, dates, or deadlines with abstract time-pressure descriptors (e.g., "an imminent time-bounded review").
   - Replace industry-specific jargon, tool names, or systems with generic terms (e.g., "a documentation page", "a tooling capability").
   - Replace specific metrics or values with abstract terms (e.g., "inconsistent key figures", "a material differential").
4. **Preserve structural elements**: keep elements that define the pattern's identity (conflict type, time pressure, role dependencies, observable problem state); abstraction strips domain specifics, not structure.
5. **The pattern must be testable against varied scenarios**: if a different domain with the same structural bottleneck would not match, abstract further.

Output format:

```
{
  "template_name": "Short descriptive name for this pattern",
  "pattern":       "Brief description of the bottleneck situation",
  "evidence_flow": ["What to check first",
                    "What to check next", ...]
}
```

Instruction constraint: respond only with the JSON object.

Figure 8: Template construction prompt for the workspace setting. Given one solved example, the model returns a reusable template added to the template pool $\mathcal{T}$. Only `pattern` and `evidence_flow` are exposed to the agent at inference.

Template Construction Prompt (Code).
You extract reusable, repo-agnostic bug patterns from solved code-bug examples.
The gold patch defines correct behavior: the pre-patch code is the bug shape, the post-patch code is the fix shape, and the diff encodes the intent.

Solved example shown to the model:

```
Repository: {repo}

Issue / Bug Report:
{bottleneck_description}

Buggy Function(s):
### {func.qualname} (file: {func.path})
```python
{func.content}
```
...

Gold Patch (the fix):
```diff
{patch}
```
```

Rules:

1. **Pattern and evidence_flow: abstract**. Replace specific identifiers with role descriptors. Both fields must transfer to a different Python codebase carrying the same *structural* bug.
2. **Code-centric detection**: each step in `evidence_flow` must reference observable code signals visible in the *buggy code alone*. The gold patch is a reference for understanding the bug, but `evidence_flow` must work without seeing the patch.
3. **One pattern per template**.
4. **Broad applicability**: pattern and evidence_flow must describe a bug shape that could plausibly appear in multiple scenarios (e.g., parsers, serializers, validators, builders, schedulers). Aim for patterns where at least three different bug scenarios outside this example would still match.

Output format:

```
{
  "template_name": "Short descriptive name (abstract)",
  "pattern":       "Abstract: what shape of code carries the bug
                    and what input/condition triggers it.",
  "evidence_flow": ["Abstract: what to check first when scanning
                     a function for this pattern",
                    "Abstract: what to check next", ...]
}
```

Instruction constraint: respond with the JSON object only.

Figure 9: Template construction prompt for the code setting. Given one solved bug-fix example, the model returns a reusable bug-pattern template added to the template pool $\mathcal{T}$. Only `pattern` and `evidence_flow` are exposed to the agent at inference.

Inference Prompt (Workspace).

Role definition. You are a proactive agent whose goal is to serve a user, about whom you will receive information in the "World Model" section. You will be provided personal context about the user, their relationships, their organizational roles and structures, their personal pain points and priorities, and the actions they are able to take. Your task is to find and identify (without prior prompting) a singular bottleneck in their lives, and recommend a resolution by selecting one of the pre-specified actions. The bottleneck must be identified by analyzing the provided documents, emails, and calendar events.

Thought templates definition. The following reasoning templates describe common bottleneck patterns. Use these as guides when analyzing documents: match observed situations to relevant templates to select the right action and fill parameters correctly. Not every template will be relevant.

Bottleneck definition. A bottleneck is a concerning issue that

- represents an important pattern or situation requiring attention (based on the World Model);
- is directly resolvable through one of the provided user actions;
- would be flagged by a competent assistant as needing intervention;
- is directly indicated within a document or set of documents in the datastore.

Previously found bottlenecks (included only after the first iteration). The following bottlenecks have already been identified in previous iterations. Do not repeat these; find only a new bottleneck that is different from those listed. If no more new bottlenecks remain, return an empty JSON object `{}`.

Task execution.

1. Analysis: examine all provided documents, emails, and calendar events to understand the user's current situation.
2. Pattern recognition: identify concerning patterns that match the bottleneck definition.
3. Action selection: choose the most appropriate action from the available actions list.
4. Response generation: provide the analysis and recommendation in the specified JSON format.

Output format.

```
{
  "bottlenecks": [
    {
      "used_template_id":   "TID_X",
      "used_template_name": "...",
      "why_matched": "Which observations in the documents satisfy
                      the matched template's pattern, with doc IDs.",
      "retrieved_documents": ["doc_id_1", "doc_id_2", ...],
      "bottleneck": "Natural-language description of the issue.",
      "action": {
        "function_name": "action_function_name",
        "parameters":    {"parameter_name1": "value1", ...}
      }
    }
  ]
}
```

The `bottlenecks` array contains exactly one identified bottleneck per iteration.

Context shown to the model:

```
Persona:     {persona}
World Model: {world_model}

Bottleneck Pattern Templates:
[{template_id}] {template_name}
  Pattern: {pattern}
  Evidence flow:
    - {step_1}
    - {step_2}
    ...

Documents:         {data_sources}
Available Actions: {available_actions}
```

Figure 10: Per-iteration inference prompt for the workspace setting. At each iteration the model receives the persona, world model, retrieved documents, available action library, current template pool, and the bottlenecks already returned in previous iterations; it returns one new bottleneck and the chosen action, or an empty object to terminate.

Inference Prompt (Code).

Role definition. You are a proactive code-maintenance agent whose goal is to surface issues in a Python software repository. Your task is to find and identify (without prior prompting) distinct issues worth reporting in the codebase, and recommend a resolution for each by producing a unified diff patch.

Bug pattern templates definition. The following templates describe bug patterns observed in past, solved examples from other repositories. They are a partial library of known bug shapes, not an exhaustive catalog of every issue this codebase may contain. Report two kinds of issues:

1. Issues that match one of the templates below. Set `used_template_id` to the template id and explain the match in `why_matched`.
2. Issues that do not fit any template but are still genuine bugs grounded in the code. Leave `used_template_id` as `""`.

Both kinds count equally.

Bottleneck definition. An issue is a problem in this codebase that a user, maintainer, or contributor would reasonably report. Each issue must

- be resolvable by editing one or more of the provided Python functions;
- point to specific function(s) and a specific change;
- be supported by concrete evidence drawn from the code itself;
- be distinct from other issues in the list (do not split one bug across multiple entries).

Previously found bottlenecks (included only after the first iteration). Report only new issues distinct from those already returned, covering both template-matched and template-novel kinds. If no new issues remain, return an empty JSON object `{}`.

Task execution.

1. Analysis: examine all provided functions to understand what each one is supposed to do.
2. Issue identification: identify problems matching the issue definition, skipping any already reported.
3. Patch generation: produce a unified diff patch resolving each identified issue.
4. Response generation: provide analysis and patches in the specified JSON format.

Output format.

```
{
  "bottlenecks": [
    {
      "used_template_id":   "TID_X" or "",
      "used_template_name": "..."   or "",
      "why_matched": "Which observations in the functions satisfy
                      the matched template's evidence_flow, citing
                      specific function IDs and lines. Use \"\"
                      if no template applies.",
      "retrieved_documents": ["func_id_1", "func_id_2", ...],
      "bottleneck": "Natural-language description, citing concrete
                     code evidence and qualnames.",
      "action": {
        "patch": "<unified diff text fixing the issue,
                  starting with `diff --git a/<path> b/<path>`>"
      }
    }
  ]
}
```

Context shown to the model:

```
Repository: {repo}

Bug Pattern Templates:
[{template_id}] {template_name}
  Pattern: {pattern}
  Evidence flow:
    - {step_1}
    - {step_2}
    ...

Functions: {data_sources}
```

Figure 11: Per-iteration inference prompt for the code setting. At each iteration the model receives the repository name, current template pool, candidate Python functions, and the bottlenecks already returned in previous iterations; it returns new bottlenecks (template-matched or template-novel) with unified-diff patches, or an empty object to terminate.