2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE)

Watson: A Cognitive Observability Framework for the Reasoning of LLM-Powered Agents

Benjamin Rombaut*, Sogol Masoumzadeh*†, Kirill Vasilevski*, Dayi Lin*, Ahmed E. Hassan‡
*Centre for Software Excellence, Huawei Canada
†McGill University, Montreal, Canada
‡Queen's University, Kingston, Canada

***Abstract*—Large language models (LLMs) are increasingly integrated into autonomous systems, giving rise to a new class of software known as *Agentware*, where LLM-powered agents perform complex, open-ended tasks in domains such as software engineering, customer service, and data analysis. However, their high autonomy and opaque reasoning processes pose significant challenges for traditional software observability methods. To address this, we introduce the concept of *cognitive observability*—the ability to recover and inspect the implicit reasoning behind agent decisions. We present Watson, a general-purpose framework for observing the reasoning processes of fast-thinking LLM agents without altering their behavior. Watson retroactively infers reasoning traces using prompt attribution techniques. We evaluate Watson in both manual debugging and automated correction scenarios across the MMLU benchmark and the AutoCodeRover and OpenHands agents on the SWE-bench-lite dataset. In both static and dynamic settings, Watson surfaces actionable reasoning insights and supports targeted interventions, demonstrating its practical utility for improving transparency and reliability in Agentware systems.**

***Index Terms*—Observability, Foundation models, Large Language Models, AIware, FMware**

I. Introduction

Agentic software powered by large language models (LLMs) [6, 23, 36] (i.e., *Agentware* [19]) is increasingly being adopted across a variety of domains such as autonomous software engineering [26, 51], customer support [18], and data analytics [17]. Unlike traditional software systems, Agentware operates with high autonomy, making decisions through opaque and implicit reasoning processes executed by the underlying LLMs. While this enables greater flexibility and task generalization, it also introduces new challenges in observability, as developers can no longer rely on code-level traceability to understand or debug system behavior [20, 49]. As a result, ensuring the reliability, explainability, and controllability of such systems requires fundamentally new approaches to observing and interpreting their internal decision-making processes.

Traditional observability approaches such as logging, tracing, and performance monitoring are effective for deterministic software because developers can instrument code to trace execution and diagnose issues [11, 31, 41]. However, these techniques fall short in the context of Agentware, where behavior is driven by non-deterministic prompt-based reasoning inside LLMs. As these models tend to not expose their internal decision paths, developers are often left without insight into why an agent chose a particular action or produced a specific output. To address this gap, we propose a new concept called *cognitive observability*, which extends observability beyond system performance to capture higher-level signals about an agent's behavior and decision-making. Prior work has begun to explore aspects such as *semantic feedback* and *output integrity*, but the most underdeveloped dimension remains an agent's *reasoning trace*—the implicit steps it takes to reach a conclusion. Our work emphasizes this dimension, showing how surfacing reasoning traces provides critical support for debugging, auditing, and systematically improving Agentware in dynamic, multi-agent, and safety-critical settings.

Building on prior work that has examined semantic feedback and output integrity [11], we position our contribution within a broader taxonomy of *cognitive observability*, which we define as encompassing these existing dimensions together with the underexplored notion of reasoning trace. Our emphasis is on this latter dimension: recovering the implicit steps an agent takes to reach its conclusions. To this end, we design and implement Watson, a general-purpose framework for observing the reasoning processes of fast-thinking LLM agents without modifying their behavior. We evaluate Watson in both manual debugging and automatic correction scenarios across the Massive Multitask Language Understanding (MMLU) [22] benchmark and the AutoCodeRover [51] and OpenHands [44] agents on the SWE-bench-lite [5] benchmark. In both static and dynamic task settings, reasoning traces generated by Watson surface actionable insights and enable targeted interventions, demonstrating its utility both as a manual and an automated observability tool for LLM-powered systems.

The remainder of the paper is structured as follows: Section II discusses observability for Agentware. Section III introduces Watson, our proposed framework. Section IV provides a case study on applying Watson to agentic systems to demonstrate its practical applicability. Section V discusses the practical applications and limitations of Watson and the threats to our study. Finally, Section VI concludes the paper.

II. Observability for Agentware

In this section, we discuss key forms of observability relevant to Agentware, including operational and cognitive observability, and introduce the challenge of recovering reasoning traces in fast-thinking agents.

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00067

A. Operational Observability

Traditional observability in software systems, rooted in metrics, logs, and traces, has long served to ensure system reliability, diagnose failures, and track performance regressions [31]. This paradigm has extended to Agentware, where operational observability focuses on surface-level telemetry such as token usage, latency, and system call outcomes [11]. In agent systems that coordinate multiple model inferences and interact with external components, such observability remains essential for debugging low-level failures and ensuring workflow stability. Several industry and research efforts have contributed observability tooling specifically tailored to Agentware pipelines [2, 13, 21, 29, 30, 35, 39, 43, 47], offering visibility into system bottlenecks and usage patterns.

However, while operational observability can highlight where and when a failure occurs, it provides little insight into why an LLM-powered agent produced a particular response, especially when that response reflects reasoning errors, subtle misunderstandings, or emergent behaviors not grounded in explicit logic. In such cases, simple input-output traces are insufficient for understanding the underlying cognitive processes of LLM-based agents. This gap motivates the need for cognitive observability, a complementary layer of observability that seeks to expose the internal reasoning and decision-making dynamics of agents rather than just their surface behaviors.

B. Cognitive observability

Where operational observability focuses on system performance and resource usage, we propose the notion of cognitive observability to capture higher-level aspects of an agent's behavior and decision-making that are often not reflected in traditional telemetry. Under this umbrella, cognitive observability can draw on a variety of signals that shed light on how an agent processes information, generates outputs, and its performance is evaluated. Some of these ideas have already been explored in prior work [11]. For example, semantic feedback, which captures explicit, implicit, and freeform user reactions (e.g., ratings, textual feedback, or usage patterns) [16, 37], and output integrity, which analyzes the agent's outputs to assess qualitative properties such as correctness, hallucinations, or sentiment [24]. Both of these signals, as well as others, have begun to be supported by a growing ecosystem of tools [2, 25, 29, 35, 48]. In contrast, our work emphasizes a less developed aspect of cognitive observability: the recovery of an agent's reasoning trace.

Among the different signals encompassed by cognitive observability, reasoning trace remains the most underdeveloped, which we define as the implicit cognitive steps an agent takes to arrive at a decision. Unlike semantic feedback or output integrity, which can often be observed directly through interactions with users or inspection of outputs, reasoning traces are rarely exposed in fast-thinking LLM agents. Yet, understanding how an agent reached a particular conclusion is crucial for diagnosing and addressing errors, especially in complex Agentware systems where multiple agents interact

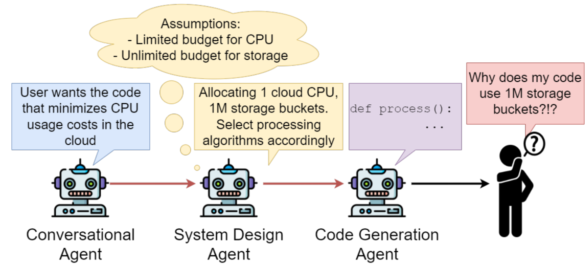

Fig. 1. In multi-agent systems, errors made by early agents may only manifest later in the workflow.

and coordinate with each other and with external components. In such settings, mistakes made early in the workflow can propagate and only manifest much later, making it difficult to localize their source. As illustrated in Figure 1, erroneous assumptions introduced by one agent may ripple downstream and surface as failures in later stages. Cognitive observability through reasoning traces is therefore essential: it provides developers with a window into the decision-making process, enabling them to identify why an agent behaved as it did, trace the propagation of errors, and more systematically debug failures in complex multi-agent systems.

C. Reasoning Trace Recovery in Fast-Thinking Agents

Building on this motivation, we next consider the practical challenge of recovering reasoning traces in fast-thinking LLM agents. Unlike LRMs [9, 38], which explicitly generate chains of thought but incur high latency and cost [46, 50], agents driven by fast-thinking LLMs do not provide the same level of transparency in their decision-making process. These models generate outputs through a token-by-token sampling process, without producing an internal "transcript" of how they arrived at a decision. Attempts to recover a reasoning trace (e.g., prompting the model to explain its answer or to "think step-by-step" using chain-of-thought (CoT) prompting) inevitably change the input and alter the generation process. This interference poses a fundamental challenge to observability: the act of asking for a justification can modify the very behavior we aim to observe. Recent work has shown that CoT prompting can reduce performance on tasks that benefit from fast, intuitive responses. For example, Liu et al. [34] found that prompting models to explain their reasoning can significantly reduce accuracy on tasks requiring human intuition or commonsense reasoning, with performance dropping by as much as 36% compared to direct answering. Similarly, Su et al. [42] trained a model to switch between "fast" and "slow" modes and observed that CoT reasoning led to substantially different completions.

Beyond accuracy differences, there is growing evidence that post-hoc explanations are often unfaithful. Chen et al. [8] demonstrate that models exposed to subtle prompt cues would change their answers accordingly, but omit any mention of the cue when asked to justify their output. Liao and Vaughan [33] further showed that models often ignore influential prompt features in their explanations, suggesting that self-generated reasoning traces may not reflect the actual decision path taken. Together, these findings highlight a central technical challenge in agent observability: recovering an agent's reasoning trace without modifying its original behavior or inducing hallucinated justifications.

To address these challenges, we introduce a novel approach for retrospectively observing the implicit reasoning of fast-thinking agents without interfering with their original behavior. Rather than prompting the agent to articulate its reasoning during generation, which can distort the output, we instead recover plausible reasoning traces after the fact by mirroring the agent's configuration and constraints. Our focus is not on improving the quality of the output itself, but on faithfully reconstructing the cognitive steps that could have led to it. To ensure these recovered traces are not arbitrary or hallucinated, we incorporate verification mechanisms such as self-consistency checks across multiple reasoning paths and prompt attribution techniques to confirm that the explanations are aligned with the agent's original prompt and completion. This approach forms the foundation of our proposed framework, introduced in the next section.

III. Watson: A Framework to Observe Reasoning Traces of LLM-powered Agents

In this section, we introduce Watson, a framework designed to observe the reasoning traces of LLM-powered agents without interfering with their behavior (Figure 2).

A. Overview

In the Watson framework, a "surrogate agent" [32] operates in parallel with a "primary agent" (i.e., the agent under observation). The primary agent generates outputs that conform to the constraints and expectations of the system it operates in, and our goal is to observe its behavior without altering it. In contrast, the surrogate agent reproduces the primary agent's output while also generating a detailed, step-by-step account of the reasoning process that could have led to that outcome. This reasoning trace is then verified to assess its fidelity to the primary agent's actual generation process. By decoupling reasoning from action, Watson enables developers to recover the implicit reasoning path of the primary agent without affecting its behavior. The resulting explanations provide valuable insights into the agent's decision-making process, offering interpretable signals that can be used to debug, evaluate, or improve the agent without altering its operational dynamics.

Watson addresses the challenge of recovering reasoning traces from fast-thinking LLM-powered agents without altering their behavior or introducing unfaithful explanations. It is built around three key ideas. First, the surrogate agent closely mirrors the primary agent to replicate its behavior and ground its reasoning in the same decision process. Second, the surrogate generates reasoning paths linking the input prompt to the primary agent's output, providing coherent explanations without affecting the original generation. Third, Watson checks that the surrogate's reasoning aligns with the actual

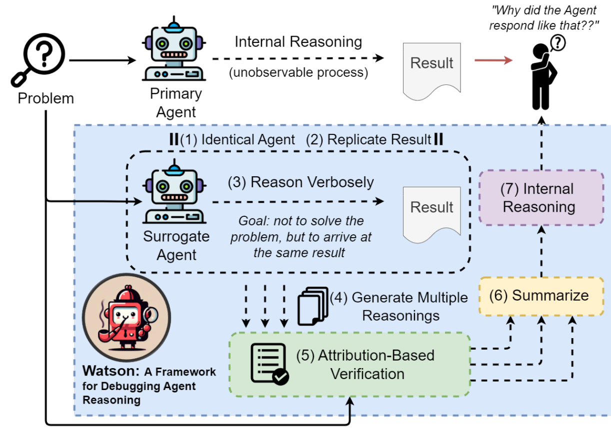

Fig. 2. Overview of Watson, a framework to observe the reasoning process of agents without affecting their behavior.

prompt components that influenced the primary agent's output. These validated traces are then summarized into a higher-level "meta-reasoning", akin to self-consistency [45], to capture the agent's implicit logic. We detail each of these considerations below.

B. Mirroring Configuration of Primary Agent

To accurately recover the primary agent's implicit reasoning, the surrogate agent must mirror its configuration as closely as possible, most critically in terms of the underlying LLM architecture and decoding parameters. Using the same language model ensures that both agents interpret inputs with equivalent linguistic and reasoning capabilities. Equally important are decoding parameters such as *temperature* and *top_p*, which control the stochasticity and diversity of generated outputs; even small differences in these settings can produce significant divergence in behavior. While configuration mirroring is necessary for behavioral alignment, it is not sufficient to guarantee output equivalence due to inherent non-determinism in LLM decoding and sensitivity to initialization [7]. We address this challenge in the final component of the Watson framework (Section III-D).

C. Generating Reasoning Paths

The goal of Watson is to use a surrogate agent to generate the implicit reasoning path the primary agent's input prompt to its final output. To achieve this, the surrogate agent is given access to the primary agent's input prompt and the corresponding generated output, and is tasked with producing a reasoning path that potentially bridges the two. However, this introduces a novel design challenge, as most autoregressive LLMs are trained for left-to-right generation, predicting the next token based solely on previously seen context. As a result, they lack an inherent ability to generate coherent reasoning that fills in the middle between a fixed prompt and completion.

To address this challenge, we adopt a technique known as *fill-in-the-middle* (FIM) [4], which enables decoder-only language models to generate text that fits between a given

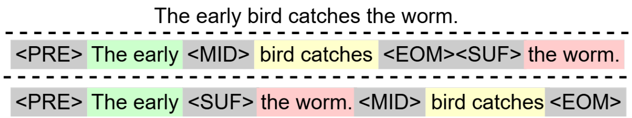

Fig. 3. Fill-in-the-Middle (FIM) prompt example. Top: original prompt; middle: with FIM tokens; bottom: prefix-suffix-middle format used for training.

beginning and end. Originally developed for code completion tasks[14, 27], FIM allows the model to condition on both a *prefix* and a *suffix* when generating content. The input is segmented using special tokens: <PRE> for the prefix, <MID> and <EOM> to mark the region to be generated, and <SUF> for the suffix. During training, the model learns to complete the middle section based on the surrounding context, effectively learning to bridge the gap between an input and its corresponding output (Figure 3).

To implement FIM in Watson, the surrogate agent is prompted with the primary agent's input as the *prefix* and the primary agent's output as the *suffix*, and is tasked with infilling a plausible reasoning path between them. To better elicit structured reasoning, we lightly augment the original prompt by appending the phrase "*Let's think step-by-step before arriving at the answer.*" to the prefix, and prepending "*Therefore, I think the answer is*" to the suffix. This approach leverages the model's ability to condition on both the beginning and end of a sequence, encouraging it to generate a coherent rationale that links the given input to the observed output. By structuring the prompt in this way, Watson enables the surrogate agent to produce a potential explanation of how the primary agent may have arrived at its response.

While FIM is well suited for generating reasoning paths, its use is limited to LLMs that natively support the FIM mechanism and can perform infilling between a given prefix and suffix. In cases where the primary agent relies on an LLM that has not been trained with FIM capabilities, we propose an alternative technique called "Repetitive Chain-of-Thought" (RepCoT). Unlike FIM, RepCoT is compatible with any LLM by prompting the surrogate agent with the same input as the primary agent, augmented with an explicit instruction to generate CoT reasoning [46]. To preserve alignment with the primary agent's behavior, any generations that do not conclude with the same output as the primary agent are discarded. To determine whether the surrogate's output matches the primary agent's, we use an LLM-based judge to assess whether the two outputs are semantically equivalent. This step is crucial in RepCoT as it ensures that the reasoning path remains consistent with the original decision, capturing only relevant explanations. However, this process may result in higher inference costs due to the need for multiple generations if we must discard generated reasonings that lead to a different outcome from the primary agent's original output.

D. Verification of the Reasoning Consistency

To ensure that the surrogate agent's reasoning accurately reflects the primary agent's implicit decision process, it is important that the generated reasoning aligns with the original inference of the primary agent that produced the final output. To support this, we first task the surrogate agent with generating a diverse set of potential reasoning paths. This population-based approach helps capture multiple plausible lines of reasoning that could have led to the same answer, offering a broader and more complete view of the primary agent's underlying thought process.

To address the risk of misaligned explanations, we validate whether each reasoning trace generated by the surrogate agent genuinely reflects the primary agent's original inference process. This step is crucial, as prior work (see Section II-C) has shown that LLMs can omit influential prompt features or hallucinate justifications when asked to explain their outputs [1, 33]. To perform this validation, we apply PromptExp [12], a framework for prompt explainability that measures the attribution of different prompt components (i.e., how much each part of the input contributed to the final output). The core idea of applying this framework in Watson is to verify that if a particular line of reasoning appears in a surrogate-generated trace, the corresponding component in the original prompt contributed meaningfully to the primary agent's output.

The first step in validating a surrogate-generated reasoning trace is to decompose the input prompt into semantically meaningful components. There are multiple ways to perform this decomposition depending on the type of task being considered. For natural language tasks, the prompt can be split into sentences or clauses; for code tasks, decomposition may be performed by functions or logical blocks; for dialogue tasks, prompt components can correspond to individual speaker turns. Choosing the appropriate granularity is critical because when components are too coarse, subtle but influential elements may be obscured, which can lead to underestimation of their contribution. Conversely, when components are too fine, the resulting attribution scores may be noisy or fragmented, reducing the reliability of the validation process. By default, Watson decomposes prompts into sentences, newlines, and code blocks enclosed in triple backticks. To support more specialized use cases, users may also supply a custom prompt decomposer, enabling task-specific segmentation strategies that go beyond these default heuristics.

Once the prompt has been decomposed, we use PromptExp to quantify the influence of each component on the primary agent's output. Each component is individually masked to create a series of perturbed inputs, which are then passed to the primary agent to produce a new output. To estimate how much each component influenced the original output, we compare the model's response to each perturbed input with the original completion. Specifically, we calculate the shift in token probabilities between the two outputs, where a larger shift indicates that the masked component played a greater role in shaping the original response.

The resulting attribution scores are then aggregated across all components using the method defined in PromptExp. This allows us to determine which prompt features were most influential in the primary agent's inference, and in turn, to validate whether a surrogate-generated reasoning trace is grounded in the actual decision factors used by the original model.

To perform this validation, we employ an LLM-as-a-judge that explicitly evaluates whether a reasoning trace aligns with the primary agent's influential prompt components. The judge is provided with the original prompt, the surrogate reasoning, and the top-n and bottom-n ranked prompt components based on attribution scores, where n is user-configurable, with a default of 10. We select 10 as it typically covers all components in small- to medium-sized prompts, while for larger prompts it provides a representative subset without overwhelming the judging model with excessive context. In our implementation, the judge is instructed to determine whether the reasoning emphasizes the high-attribution components while not disproportionately focusing on the low-attribution ones, and to respond with a binary output: "YES" if the reasoning aligns appropriately or "NO" if it does not. Reasoning traces that receive a "NO" from the judge are discarded, and new traces are generated until a configurable population size of aligned reasonings is obtained. This setup also allows for flexibility: users can adjust the number of top and bottom components considered or modify the judge instructions to reflect task-specific evaluation criteria. A simplified example illustrating this reasoning alignment process, including the prompt components, their attribution values, the surrogate reasoning, and the validation results, is shown in Table I.

TABLE I
ILLUSTRATIVE EXAMPLE OF REASONING ALIGNMENT WITH EXPLICIT INSTRUCTIONS AND NUANCED ATTRIBUTION ACROSS A MULTI-PART PROMPT.

Prompt (Attribution)	*Please evaluate each action below and indicate which is morally wrong.* ***(0.10)*** *1. I helped a neighbor fix their car.* ***(0.10)*** *2. I took credit for a colleague's project at work.* ***(0.55)*** *3. I told a white lie to avoid hurting someone.* ***(0.25)***
Response	*The second action is morally wrong*
Attribution Analysis	The model's output is predominantly influenced by the phrase "*2. I took credit for a colleague's project at work.*" with the highest attribution (0.55), and the third phrase being somewhat influential (0.25). The instruction line and first phrase have smaller attributions (0.10 and 0.10).
Reasoning	*Helping a neighbor fix their car is a kind and altruistic act and therefore not morally wrong. Taking credit for a colleague's project is dishonest and violates ethical standards, making it morally wrong. Telling a white lie to avoid hurting someone's feelings is generally viewed as socially acceptable and thus not morally wrong.*
Validation	The surrogate reasoning appropriately emphasizes the high-attribution dishonest action and reasonably addresses the other parts, demonstrating good alignment with the model's attribution and output.

Finally, we summarize the reasoning traces that passed verification to extract their mutual threads and recurring ideas, similar to Wang et al. [45], who explored identifying shared reasoning across multiple explanations. This step produces a concise explanation that reflects the primary agent's most likely decision path, while still preserving meaningful variation across the aligned traces. By emphasizing points of agreement across the population, the summary highlights reasoning patterns that are both plausible and consistent with the agent's original behavior.

To generate this summary, we employ another LLM acting as a reasoning summarization agent. The agent is provided with the full set of verified reasoning traces as a structured list and instructed to synthesize them into a single, coherent reasoning path. The prompt explicitly directs the agent to capture the most frequently occurring reasoning patterns, maintain logical coherence, avoid contradictions, and produce a natural, standalone explanation without introducing any new ideas or meta-summary language. This approach ensures that the resulting summary reflects consensus among the verified traces while remaining faithful to the original reasoning processes.

IV. CASE STUDY

This section presents a case study illustrating how the Watson framework enables semantic observability in LLM-powered agentic systems. Rather than evaluating or extending Watson itself, we use it as a diagnostic tool—analogous to a debugger in traditional software development. We begin with an example of Watson-generated reasoning and a preliminary manual study in which a developer uses reasoning traces to diagnose real-world failures in the AutoCodeRover system. This demonstrates the interpretability and practical utility of Watson 's outputs for fault localization.

We then show how Watson-generated can support automated debugging pipelines across two domains: multiple-choice question answering and autonomous software engineering. Together, these studies demonstrate that semantic observability enables both human and agentic insight into model reasoning. The remainder of this section outlines the purpose of the case study, describes the experimental setup, and presents results highlighting the value of incorporating reasoning traces into agent workflows.

A. Purpose and Positioning

This case study demonstrates the practical utility of the Watson framework by applying it to agentic systems and using the Watson-generated reasoning traces to both manually and automatically debug the performance of a LLM-powered agent. Importantly, this case study does not extend the Watson framework itself. Rather, it builds on top of Watson as a semantic observability tool, illustrating how the reasoning traces it produces can be used both by human developers and in an external, automated debugging pipeline.

The goal of this case study is not to directly enhance the reasoning capabilities of the underlying LLM or to optimize the accuracy of the primary agent. Instead, we draw a parallel to traditional software debugging workflows, where tools such as the GNU Debugger (GDB) [15] are employed to observe and diagnose faults in program execution, without being expected to fix or improve the software themselves. Similarly, Watson enables the observation of an LLMs latent reasoning traces, which can then be analyzed or acted upon by a human or another agent.

To support this demonstration, we first conducted a preliminary manual debugging study, in which we analyze Watson-generated reasoning traces to identify failure modes in agent behavior. In our analysis, the traces revealed latent reasoning errors and misaligned assumptions that were not evident from the final answers alone. Building on these findings, we introduce an automated debugging agent that consumes the same reasoning traces to detect and intervene on faulty reasoning paths. This agent operates independently of both the primary and surrogate agents, demonstrating how Watson facilitates semantic observability without modifying the underlying model or agents.

Although we report metrics such as pass@k improvement to quantify the outcomes of this automated debugging process, these results should be interpreted primarily as evidence of the practical utility of semantic observability. Rather than evaluating Watson's ability to directly optimize model reasoning, our goal is to illustrate how external systems, either human or automated, can effectively leverage reasoning traces produced by Watson to enhance transparency and support debugging workflows.

B. Task Selection and Study Design

We evaluate the utility of Watson on both non-agentic and agentic tasks to demonstrate its generalizability and effectiveness across domains. The first task involves the MMLU benchmark, a diverse set of multiple-choice questions spanning subjects such as professional law, high school psychology, and moral reasoning. From the full set of 14,042 questions, we randomly sample three subsets totaling 374 questions each, which corresponds to the minimum required for statistical significance at a 95% confidence level with a 5% margin of error. These tasks serve as a controlled environment for observing LLM behavior outside of agentic workflows.

As a first step, we present a qualitative example from MMLU to illustrate the kinds of reasoning Watson can produce. In Listing 1, we show an incorrectly answered question from the professional law category, where the agent selects option *B* by focusing on the offer's duration, rather than the correct option *A*, as well as the corresponding reasoning trace generated by Watson. The trace (1) accurately reflects the primary agent's chosen answer, (2) considers the merits of alternative options, and (3) articulates a coherent rationale for the agent's decision. This example highlights Watson 's ability to produce detailed and logically grounded reasoning even in static, non-agentic tasks.

Listing 1. Example MMLU benchmark question, the initial incorrect answer, and the associated reasoning trace generated by Watson

```
You are a general knowledge quiz master. You are going to
 answer a question from a quiz. Only respond with the
 correct answer.
[Question] A merchant's irrevocable written offer (firm
 offer) to sell goods
[Options]
A) Must be separately signed if the offeree supplies a form
  contract containing the offer.
B) Is valid for three months.
C) Is nonassignable.
D) Can not exceed a three-month duration even if
 consideration is given.
[Answer] B) Is valid for three months.
[Reasoning] The question is asking for the characteristics
 of a merchant's irrevocable written offer to sell goods.
Option A mentions that the offer must be separately signed
 if the offeree supplies a form contract containing the
 offer. This statement does not directly answer what is
 being asked in the question.
Option B states that the offer is valid for three months.
 This seems like it could be a possible answer, as it
 specifies a duration for which an offer can be considered
  valid.
Option C states that the firm offer is nonassignable. While
  this may or may not be true, it does not pertain to the
 characteristics of a merchant's irrevocable written offer
 .
Lastly, option D states that such an offer cannot exceed
 three months even with consideration given. Again, while
 this may or may not be true, it does not directly address
  what makes up a merchant's irrevocable written offer to
 sell goods.
Based on these options and considering all factors involved
  in answering this question correctly, I believe option B
  - Is valid for three months - would make sense as an
 appropriate response because: (1) It specifically
 addresses one characteristic of such an offer (validity
 period), (2) It doesn't mention anything outside of what
 was asked, (3) The other options don't seem relevant
 enough
```

To evaluate Watson in more complex, real-world settings, we incorporate AutoCodeRover, an autonomous software engineering agent that resolves GitHub-style issues by analyzing and modifying source code. We apply AutoCodeRover to SWE-bench-lite, a benchmark of 300 issues derived from the broader SWE-bench dataset [28]. AutoCodeRover operates in two main phases: first, it retrieves and synthesizes relevant context from the codebase to identify likely faulty regions; second, it generates code patches to fix the issue. Because overall performance hinges on the agent's initial code localization step, our evaluation focuses exclusively on this first, opaque phase.

To assess generalizability across agent scaffolds, we additionally evaluate the OpenHands on the same SWE-bench-lite benchmark. This allows us to observe whether Watson can similarly capture latent reasoning traces and guide debugging across different agent architectures and underlying models.

To gain qualitative insight into Watson 's effectiveness in this setting, we conduct a manual debugging analysis on two failure cases from the AutoCodeRover evaluation. A human developer reviews the issue report, the original AutoCodeRover output, and the Watson-generated reasoning trace (Section III) to diagnose potential missteps in the agent's reasoning. Based on these insights, the developer then crafts a targeted decision-refining hint, which is fed, along with the original prompt, back into AutoCodeRover to reattempt the issue. This manual study helps assess whether Watson's traces can meaningfully aid human understanding and correction of complex agent behavior.

Building on these qualitative findings, we conduct a full-scale automated study using dedicated debugging agents on MMLU, AutoCodeRover, and OpenHands. In each case, we generate hints from the reasoning traces and inject them into the agent's prompt to guide its decision-making. For

Listing 2. The Watson-generated reasoning for instance `django__django-13401`, highlighting key reasoning steps. The agent incorrectly focuses on `model_inheritance/tests.py`, which contains subclasses `B` and `C`, and proposes changes there, rather than targeting the actual source of the bug.

```
The common points across the provided explanations can be summarized as follows:
1. **Identified Bug Location**: All reasonings consistently highlight that the issue is centered around the
 model_inheritance/tests.py file , which contains classes A, B, and C. This file is repeatedly mentioned as critical to
 understanding where potential bugs may lie.
2. **Focus on Classes B and C**: There is a shared emphasis on examining classes B and C due to their inheritance from
 class A, suggesting that any issues likely originate from these subclasses or the abstract base class.
3. **Method Investigations**: Several explanations point out specific methods (`__eq__`, `__lt__`, and `__hash__`) in
 relation to fields within these classes, indicating a need to analyze how equality comparisons are implemented for
 instances of these models.
4. **Sufficient Context Established**: Multiple reasonings express confidence that enough context has been gathered
 regarding relevant files and classes without needing further information at this stage.
```

Listing 3. Segment of AutoCodeRover output for instance `django__django-13401` with a reasoning hint. The hint guides AutoCodeRover to the correct patch in `django/db/models/fields/__init__.py` and to rule out `tests.py` as the source.

```
[ASSISTANT] The issue reported is related to the equality comparison of two fields in different models. The author found
 that if they pull the fields of two models into a shared set, one will be de-duplicated away, because they compare as
 equal. The root of the issue is that they compare equal, as fields only consider self.creation_counter when comparing
 for equality. So, we need to adjust __eq__ so that if the field.model is different, they will compare unequal.
 Similarly, it is probably wise to adjust __hash__ and __lt__ to match. When adjusting __lt__, it may be wise to order
 first by self.creation_counter so that cases not affected by this equality collision won't be re-ordered. In the
 experimental branch, there was one test that broke if ordered them by model first.
The fix for this issue is not located in any tests.py files , although they are a good place to start to get more
 context on the issue. We also found that the code snippets we searched for are mostly located in the file
 django/db/models/fields/__init__.py , which makes sense as this is where the Field class is defined. We can also see
 that the class A is inherited by classes B and C, and B and C are inherited by other classes in the codebase. This
 could be the reason why the issue is not located in any `tests.py` files, as it may be related to the inheritance of
 classes. Based on the context we have gathered, we can now start writing patches to solve the issue.
```

MMLU, we evaluate the impact of these hints on answer accuracy, while for both AutoCodeRover and OpenHands, we assess their effect on localization performance across the entire benchmark.

C. Manual Debugging with Reasoning Traces

To explore the practical utility of Watson in real-world debugging scenarios, we conducted a manual analysis using two failure cases from the AutoCodeRover evaluation set (i.e., cases where the incorrect code files were identified by the first agent as buggy locations). For each failure case, we began by presenting the corresponding issue report and the output of running AutoCodeRover on the task to a human developer, along with the associated reasoning trace generated by Watson (Section III). The developer then attempted to infer which aspects of the issue were emphasized or overlooked by the agent and whether these misalignments could plausibly explain the failure. Based on this analysis, we constructed a decision-refining hint and provided it, alongside the original input prompt, to AutoCodeRover in an attempt to re-solve the issue.

When executing these manual cases, we use the FIM configuration of Watson, and use GPT-3.5-turbo-instruct as the underlying model, as it is the only OpenAI model that explicitly supports the FIM capabilities required. In this setup, both the primary and surrogate agents share the same model and decoding parameters to ensure consistency. For reasoning validation, we employ the default prompt decomposer and set $n = 10$, meaning that the judge considers up to the 10 most- and 10 least-attributed prompt components to verify that the surrogate reasoning aligns with the aspects the agent attended to. We generate a minimum of 10 reasoning traces per task. To summarize the resulting reasoning populations and perform related analysis tasks, we use OpenAI's GPT-4o-mini model, selected for its improved reasoning quality and reliability.

1) Instance django__django-13401: [1] This issue, reported in the Django [10] web framework, concerns a bug where attribute fields from different models—each inheriting from the same parent model—are incorrectly treated as equal. The problem arises because field equality is determined solely by the `self.creation_counter` attribute. As a result, fields from models `B` and `C`, both of which inherit from model `A`, are considered equal despite belonging to different subclasses. This leads to unexpected behavior—for example, when fields from models `B` and `C` are added to a set, one may be erroneously deduplicated. The issue reporter proposes modifying the `__eq__`, `__hash__`, and `__lt__` methods to ensure that fields from different parent models are correctly treated as unequal.

Output of Reasoning: The reasoning generated from Watson for `django__django-13401`, shown in Listing 2, suggests the agent is focusing on the test files, specifically `model_inheritance/tests.py`, which is not the code location for where we expect the solution patch to be applied. Following this incorrect decision pathway, AutoCodeRover attempts to modify subclasses `B` and `C`, which are contained in `model_inheritance/tests.py`, as the fix for the encountered issue.

After Fixing with Hints: Based on the Watson-generated reasoning, we add the following hint at the end of the issue report and rerun AutoCodeRover.

[1] https://code.djangoproject.com/ticket/31750

Listing 4. Watson-generated reasoning for instance `django__django-14238`, showing focus on `example/core/models.py` while overlooking the issue's reference to `__subclasscheck__`, absent from all explanations.

```
**Common Points Across Reasonings:**
1. **Focus on "MyModel" and `__init__` Method**: Multiple explanations consistently mention the class "MyModel" and its
 constructor method `__init__`, indicating that they are central to identifying the bug locations.
2. **DEFAULT_AUTO_FIELD Issues**: There is a recurring theme regarding problems with `DEFAULT_AUTO_FIELD`, suggesting it
 is linked to the bugs being analyzed.
3. **File Location**: The file "example/core/models.py" is frequently identified as a potential
 source of issues, reinforcing its importance in diagnosing the problem.
```

The fix for this issue is not located in any "tests.py" files, although they are a good place to start to get more context on the issue.

With this hint, AutoCodeRover arrives at the correct file location to be patched, while specifically acknowledging that the fix is not located in any test files (Listing 3).

2) Instance `django__django-14238`: [2] This issue is reported in the Django framework and describes an application crash during model initialization when `DEFAULT_AUTO_FIELD` is set to a custom subclass of `BigAutoField`. The issue report includes a traceback demonstrating a `ValueError` requiring that the custom primary key class must subclass `AutoField`. The issue stems from a failure in `AutoFieldMeta.__subclasscheck__`, which does not recognize valid subclasses of `BigAutoField` or `SmallAutoField`. The issue report includes a suggested fix to modify the subclass check to include these subclasses in the `_subclasses` property.

Output of Reasoning: The reasoning generated from Watson for `django__django-14238`, shown in Listing 4, suggests that the agent is placing its focus on the files in the `example` directory. However, upon closer inspection of the original issue report, it is explicitly mentioned that the fix can be applied to the `__subclasscheck__` method, which does not show up at all in the reasonings, indicating that the agent is ignoring this information.

After Fixing with Hints: Based on the reasonings, we decide to add the following hint at the end of the issue report and rerun AutoCodeRover.

The fix for this issue would not be located in the 'example' directory. Make sure you focus on the '__subclasscheck__' method.

With this hint, AutoCodeRover retrieves the correct contextual information related to `__subclasscheck__` and arrives at the correct file location to be patched (Listing 5).

D. Automated Debugging Pipeline

Building on the insights gained from the manual debugging study, we design a two-stage automated evaluation pipeline to demonstrate how Watson-generated reasoning traces can be effectively leveraged in downstream debugging and decision-making processes. Rather than aiming to directly improve agent performance, this pipeline seeks to showcase the practical utility of semantic observability by comparing system behavior with and without Watson 's reasoning traces.

[2]https://code.djangoproject.com/ticket/32620

TABLE II
PASS@1 / PASS@3 ACCURACY ACROSS EVALUATION RUNS ON MMLU (374 QUESTIONS PER RUN). CONFIGURATIONS: RO-SR = RESPONSE-ONLY SELF-REFLECTION, WA-SR = WATSON-AIDED SELF-REFLECTION.

Configuration	Run 1	Run 2	Run 3
RO-SR (FIM)	59.4% / 68.2%	59.4% / 69.8%	60.7% / 71.1%
WA-SR (FIM)	58.8% / 66.8%	61.0% / 71.4%	63.3% / 73.5%
RO-SR (RepCoT)	63.6% / 70.9%	63.1% / 71.4%	66.3% / 74.3%
WA-SR (RepCoT)	61.2% / 73.0%	66.0% / 74.3%	67.9% / 77.2%

To clearly isolate the contribution of each component, we generate guiding hints for the agent under two distinct cases: (1) using Watson-generated reasoning traces along with the agent's original answer, and (2) using only the agent's original answer without any reasoning traces. This second case serves as an explicit ablation run, allowing us to measure the added value of incorporating the reasoning traces into the hint.

We begin by establishing a baseline through an ablation study. Each agentic system is first executed in its standard form (i.e., without the involvement of Watson) to produce an initial answer. For MMLU, this answer corresponds to the selected option for a benchmark question; for AutoCodeRover and OpenHands, it refers to the code location identified as faulty. We then simulate a self-reflection [40] process by prompting a separate LLM to evaluate whether the produced answer is reasonable in the context of the task (i.e., the multiple-choice question or the issue report). The reflection is performed without access to any reasoning traces, relying solely on the answer and task description. The agent is subsequently rerun on the task, with the self-reflection output injected into its prompt as a guiding hint. The correctness of the newly generated answer is then evaluated, and the result constitutes the baseline performance.

Next, we repeat the process with Watson applied to the same set of tasks. In this configuration, the surrogate agent generates a reasoning trace that explains how the primary agent arrived at its answer. This trace is incorporated into the self-reflection stage, enabling the reflection LLM to critique both the final answer and the reasoning path that led to it. As before, the resulting reflection is provided to the agent, which reruns the task with the added guidance. We then assess the correctness of this revised output. In summary, for each task, Listing 5. Segment of AutoCodeRover output for instance `django__django-14238` with a reasoning hint, enabling AutoCodeRover retrieve context on `__subclasscheck__` and locate the correct patch in `django/db/models/fields/__init__.py`, avoiding prior focus on the `example` directory.

```
[ASSISTANT] Based on the search results, we can see that there are 12 classes with the name MyModel in the codebase, but
 none of them are located in the "example" directory . This means that the issue isn't related to the "example" project.
We also found 1 method with the name __subclasscheck__, which is located in the django/db/models/fields/__init__.py file.
  This method is inherited from the AutoFieldMeta class and is used to check if a subclass is a subclass of AutoField.
Additionally, we found 3 snippets containing 'DEFAULT_AUTO_FIELD =' in the codebase. These snippets are located in the
 tests/test_sqlite.py, tests/check_framework/test_model_checks.py, and django/conf/global_settings.py files. These
 snippets are used to set the default primary key field type in the codebase.
Finally, we found 2 snippets containing '__subclasscheck__' in the codebase, both of which are located in the django/db/
 models/fields/__init__.py file. These snippets are used to check if a given subclass is a subclass of AutoField.
Based on this context, we can conclude that the issue is related to the django/db/models/fields/ __init__.py file ,
 specifically the AutoFieldMeta class and its __subclasscheck__ method. The fix for this issue would not be located in
 the "example" project, but rather in the django project itself.
```

we generate two hints: one that leverages Watson 's reasoning trace (the "hint from reasoning+answer" case) and one that does not (the ablation "hint from answer only" case).

Building on this setup, we systematically evaluate each task under both ablation (without Watson) and Watson-enabled conditions. For the MMLU evaluation, the Watson-enabled pipeline is tested under both configurations of the Watson framework: FIM, using GPT-3.5-turbo-instruct as the underlying language model, and RepCoT, using GPT-3.5-turbo-0125, which offers chat-based interaction while maintaining close behavioral similarity to GPT-3.5-turbo-instruct. For the AutoCodeRover experiment, we use the FIM configuration with GPT-3.5-turbo-instruct, while for the OpenHands experiment, we use the RepCoT configuration with Qwen3-8B. In both configurations, the primary and surrogate agents still share the same underlying model and are executed using identical decoding parameters. For reasoning validation, we also employ the default prompt decomposer and set $n = 10$, such that the judge examines the 10 highest- and lowest-attributed prompt components to verify alignment between the surrogate reasoning and the agent's attention. We generate a minimum population size of 10 reasonings. Finally, tasks within the evaluation pipeline (e.g., summarization of reasoning populations, hint generation, and grading outputs) are performed using OpenAI's GPT-4o-mini model, selected for its improved reasoning quality and reliability in downstream analysis tasks. We compute `pass@1` and `pass@3` for MMLU, and `pass@1` for AutoCodeRover and OpenHands.

E. Evaluation Results

This section presents the empirical results of applying the Watson framework across MMLU, AutoCodeRover, and OpenHands. Our evaluation compares agent performance under two conditions: (1) a self-reflection pipeline without access to Watson reasoning traces, and (2) a self-reflection pipeline enhanced with Watson-generated reasoning traces.

1) MMLU Results: Across three statistically significant evaluation runs on the MMLU benchmark (Section IV-B), we investigate whether incorporating Watson-generated reasoning traces supports the surrogate agent's self-reflection process. Rather than demonstrating uniform improvements in accuracy, our results, shown in Table II, highlight nuanced shifts in performance metrics that suggest the potential value of incorporating Watson-generated reasoning into the reflection evaluation process. These effects are observed across both the FIM and RepCoT configurations of Watson.

In the FIM setting, the baseline Pass@1 accuracy across the three runs ranges from 59.4% to 60.7%, while incorporating reasoning traces increases this to a range of 58.8% to 63.4%. Although Pass@1 performance shows marginal variance in the first experiment, the gains become more substantial in subsequent runs. The improvements are more consistently observed in Pass@3 accuracy, which increases from 68.2%-71.1% in the baseline to 66.8%–73.5% with reasoning, with each of the three runs demonstrating improvement. McNemar's test shows no significant differences (Pass@1 p=0.88,0.51,0.22; Pass@3 p=0.53,0.46,0.23), and overlapping 95% CIs indicate gains are trends, not definitive improvements.

For the RepCoT configuration, the baseline Pass@1 scores range from 63.1% to 66.3%, while the inclusion of reasoning increases them to 61.2%–67.9%, with the largest improvement observed in the third run. More prominently, Pass@3 accuracy improves from 70.9–74.3% (baseline) to a range of 73.0–77.2% when Watson-generated reasoning is used. These results indicate that Watson's explicit reasoning generation capabilities enhance the agent's self-reflective judgment, especially in settings where structured reasoning like RepCoT is applied. As with the FIM setting, McNemar's test shows no significant differences for RepCoT (Pass@1 p=0.28,0.18,0.49; Pass@3 p=0.25,0.11,0.09), and overlapping 95% CIs indicate gains are trends, not definitive improvements. However, the gains in Pass@3 suggest that reasoning traces enable the agent to better recognize partial correctness and near-miss solutions, improving its overall evaluative reliability.

2) AutoCodeRover and OpenHands Results: For both AutoCodeRover and OpenHands, we focus our evaluation on tasks where the agent successfully identified a buggy location. Each case is categorized as *correct* if the located code region matches the ground truth and *incorrect* otherwise; cases where the agent fails to identify any location are excluded from analysis. This approach allows us to assess the impact of Watson-generated reasoning on downstream debugging accuracy without being confounded by runs that terminate prematurely. Given the computational cost and practical constraints, we report `pass@1` for both agents to provide a tractable yet meaningful measure evaluation.

Across the two agentic experiments on SWE-bench-lite, we focus on tasks where the agent successfully identified potential buggy code locations. For AutoCodeRover, 103 valid tasks were analyzed: the baseline self-reflection pipeline achieved a Pass@1 of 71, while the Watson-enabled pipeline reached 75. For OpenHands, 57 valid tasks were considered, with a baseline Pass@1 of 33 and a Watson-enabled Pass@1 of 34 (Table III). In both cases, the slight improvements observed with Watson-generated reasoning were not statistically significant ($p > 0.05$), likely due to the limited sample size, and thus reflect trends rather than definitive gains in localization accuracy.

TABLE III
PASS@1 ACCURACY OF SELF-REFLECTION PIPELINES ON VALID AUTOCODEROVER AND OPENHANDS SAMPLES (SWE-BENCH-LITE). RO-SR = RESPONSE-ONLY SELF-REFLECTION, WA-SR = WATSON-AIDED SELF-REFLECTION.

Configuration	Valid Samples	RO-SR	WA-SR
AutoCodeRover	103	71 (68.9%)	75 (72.8%)
OpenHands	57	33 (57.9%)	34 (59.6%)

F. Comparison with Native Reasoning Models

To further evaluate the quality of Watson 's post-hoc reasoning traces, we compared them against traces produced natively by a reasoning-capable large language model. We selected Qwen3-8B, which provides a "thinking" mode to explicitly generate reasoning alongside answers. Using a statistically representative MMLU sample from Section IV-B, we ran the model twice per example: once with "thinking" enabled to collect native reasoning, and once with reasoning disabled to obtain only the final answer. In the latter case, we applied Watson to the input–output pair to generate post-hoc reasoning traces using the RepCoT configuration, as Qwen3-8B does not support FIM.

Each Watson-generated trace was paired with its corresponding native Qwen3-8B trace. We used an LLM-as-a-judge to assess whether the two traces were semantically equivalent, focusing on whether they followed the same logical progression, invoked similar assumptions, and reached conclusions through comparable reasoning, while ignoring superficial wording differences.

For Qwen3-8B, we compared Watson's post-hoc reasoning traces with the model's natively generated reasoning traces, conditioned on whether both runs produced the same final answer. Across the sampled MMLU tasks, the two runs produced the same answer in 307 cases (76.2%) and diverged in 67 cases (16.6%). When the answers matched, the reasoning traces were semantically equivalent in 212 cases (69.1%) and diverged in 95 cases (30.9%). Conversely, when the answers differed, the reasoning traces almost always diverged (64 cases, 95.5%), with only a small number of coincidental overlaps (3 cases, 4.5%, $p < 0.05$). While the divergence observed in roughly one-third of the matching-answer cases might initially appear as a limitation, it could in fact be advantageous: recent work has shown that natively generated reasoning traces are not always faithful to the model's actual inference process, often reflecting superficial rationalizations rather than genuine decision pathways [3, 8]. By contrast, Watson enforces consistency with the model's saliency during inference, allowing it to surface potential discrepancies between an agent's stated reasoning and its underlying behaviour. In this sense, divergences revealed by Watson may highlight cases where post-hoc reasoning provides a truer window into the model's decision-making, offering a valuable diagnostic signal for debugging and analysis.

G. Summary

This case study demonstrates the utility of Watson as a semantic observability framework for supporting both manual and automated debugging of LLM-powered agents. Across static (MMLU) and dynamic (AutoCodeRover and OpenHands) tasks, we find that Watson-generated reasoning traces consistently surface actionable signals that aid fault localization and enable targeted interventions. Our findings reinforce that Watson enables insight into agent reasoning without modifying the primary agent or underlying model. However, the debugging agents used in this study are illustrative rather than optimized, designed to validate trace utility rather than maximize performance. Moreover, Watson is an observability layer, not a reasoning optimizer, and its effectiveness ultimately depends on the interpretability and completeness of the extracted traces.

V. APPLICATIONS, LIMITATIONS, AND THREATS TO VALIDITY

In this section, we discuss potential applications of Watson in practical development settings and examine its limitations and threats to validity.

A. Applications in Practical Settings

A promising avenue for the adoption of Watson is its integration into observability platforms designed for agentic workflows. In such settings, workflow executions are typically instrumented to capture detailed traces of agent behavior, including inputs, outputs, and intermediate decision steps. Watson can be incorporated into these platforms as an on-demand reasoning reconstruction service, enabling users to request surrogate reasoning for specific agent calls when deeper insight is required. This capability extends traditional observability by augmenting traces with interpretable reasoning paths, thereby facilitating debugging, auditing, and comprehension of complex multi-agent workflows. Crucially, Watson achieves this without modifying or delaying the execution of the original agent, making it suitable for deployment in developer environments (e.g., IDEs), continuous integration and deployment pipelines, or production-grade observability dashboards. Such integration illustrates Watson's potential to complement existing software engineering practices with cognitive observability, thereby improving transparency and trust in LLM-powered agents.

B. Limitations

While the integration scenarios highlight Watson 's potential value, it is important to consider its performance trade-offs. Watson adds no runtime overhead to the primary agent, since reasoning reconstruction is applied post hoc to captured traces. However, generating multiple completions with the surrogate agent to recover diverse reasoning paths and produce a meta-reasoning summary (Section III-A) incurs computational cost. In our experiments, we generated 10 reasonings per instance to balance coverage and expense. Watson's computational cost then scales with the number of reasoning traces generated and the computation of prompt attribution scores used for trace verification, which scale with prompt length and component decomposition. Both FIM and RepCoT discard traces misaligned with these scores, regenerating until the desired number of valid traces is reached, further contributing to overhead.

Importantly, the number of surrogate completions provides a direct trade-off mechanism. Fewer completions reduce cost and latency but may miss important reasoning paths, while more completions increase coverage and confidence at the expense of computational resources. However, these computations do not affect the runtime of the observed agent, as they can be performed in parallel or post hoc. The exact cost depends on prompt complexity, the number of surrogate completions, and prompt segmentation for attribution scoring, allowing Watson to adapt to different deployment contexts, from lightweight development workflows to resource-intensive audits. Despite this flexibility, Watson cannot guarantee complete coverage of all plausible reasoning paths. Even with higher sampling, some trajectories may remain unexplored, especially for complex tasks with large search spaces. Consequently, the reconstructed reasoning should be interpreted as a representative sample rather than an exhaustive account of the agent's internal processes.

A further limitation of Watson arises from its reliance on an LLM-as-a-judge for reasoning validation (Section III-D). After PromptExp identifies the most and least influential prompt components, the judge checks that each generated reasoning trace aligns with high-attribution components without overemphasizing low-attribution ones. While this grounds validation in the agent's decision factors, the judge can misclassify traces, either rejecting plausible reasoning or accepting partially misaligned ones. We mitigate this by using a strong judge model and combining attribution scores with explicit rules (Section III-D), but some subjectivity and bias remain. The fidelity of verified traces thus depends on both surrogate coverage and judge evaluation, highlighting directions for improved calibration, robustness, and reliability in future work.

Finally, it is important to note that Watson has so far only been evaluated on text-based reasoning agents, and its applicability to other types of agents remains uncertain. Extending Watson to these domains would likely require additional mechanisms for capturing and interpreting non-textual context, as well as adapting the verification procedures to handle richer forms of agent reasoning. Consequently, the framework's generalizability beyond text-based reasoning agents should be considered a limitation and an area for future research.

C. Threats to Validity

Several factors may influence the reliability and generalizability of the experimental results reported for Watson. First, the fidelity of the surrogate agent is critical: if the surrogate does not accurately mirror the primary agent's reasoning behavior, the reconstructed traces may not reflect the true internal processes, limiting the validity of our conclusions. Second, the evaluation depends on the LLM-as-a-judge used for reasoning verification. Biases or misclassifications by this model can affect which reasoning traces are retained, potentially influencing the resulting meta-reasoning summaries. Third, the framework has only been evaluated on text-based reasoning agents, and it remains uncertain how well the results would generalize to multimodal, embodied, or highly interactive agents. Finally, reproducibility is a potential concern: variations in surrogate configurations, the number of completions generated, or the choice of judge model may yield different reasoning reconstructions, affecting consistency across experiments. By explicitly acknowledging these threats, we aim to provide a transparent assessment of the contexts in which Watson's results can be confidently interpreted.

VI. Conclusion

In this paper, we introduced the concept of *cognitive observability* as a crucial advancement in the observability of agentic software powered by foundation models. Traditional observability techniques fall short in these systems due to the opaque, non-deterministic reasoning processes of LLMs. To address this challenge, we proposed and implemented Watson, a novel framework that retrospectively recovers implicit reasoning traces with high fidelity while preserving the efficiency of standard LLMs.

Our evaluation demonstrates that Watson enables both manual debugging and automated runtime correction, effectively exposing the "why" behind agent decisions in a manner that was previously inaccessible without affecting the agent's behavior or outputs. Through empirical validation on real-world agent tasks, including MMLU, and AutoCodeRover and OpenHands on the SWE-bench-lite benchmarks, Watson facilitates improved reasoning transparency and supports enhanced reliability and controllability of Agentware. Overall, Watson offers a practical step toward addressing the observability challenges of LLM-powered agents, helping developers better interpret and improve agent behavior. Future work should explore broader generalization across agent types and tasks, more sophisticated use of reasoning traces for intervention or retraining, and real-time observability in production environments. As LLMs continue to take on complex roles in Agentware, advancing tools for cognitive observability will be critical to supporting their safe and effective deployment.

REFERENCES

[1] Anthropic Safety Team. *Reasoning models don't always say what they think*. Anthropic Technical Report. Available at https://www.anthropic.com/research/reasoning-models-dont-say-think. 2025.

[2] Arize. *Phoenix by Arize*. last accessed: 2024-10-08. 2024. URL: https://phoenix.arize.com/.

[3] Fazl Barez et al. *Chain-of-Thought Is Not Explainability*. 2025. URL: https://www.alphaxiv.org/abs/2025.02v3.

[4] Mohammad Bavarian et al. *Efficient Training of Language Models to Fill in the Middle*. 2022. URL: https://arxiv.org/abs/2207.14255.

[5] Jiayi Geng Carlos E. Jimenez John Yang. *SWE-bench Lite: A Canonical Subset for Efficient Evaluation of Language Models as Software Engineers*. last accessed: 2024-10-08. 2024. URL: https://www.swebench.com/lite.html.

[6] Yupeng Chang et al. "A Survey on Evaluation of Large Language Models". In: *ACM Transactions on Intelligent Systems and Technology* 15.3 (2024). ISSN: 2157-6904. DOI: 10.1145/3641289. URL: https://doi.org/10.1145/3641289.

[7] Boyuan Chen et al. "Towards training reproducible deep learning models". In: *Proceedings of the 44th International Conference on Software Engineering*. 2022, 2202–2214. ISBN: 9781450392211. DOI: 10.1145/3510003.3510163. URL: https://doi.org/10.1145/3510003.3510163.

[8] Yanda Chen et al. *Reasoning Models Don't Always Say What They Think*. 2025. arXiv: 2505.05410 [cs.CL]. URL: https://arxiv.org/abs/2505.05410.

[9] DeepSeek-AI et al. *DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning*. 2025. DOI: 10.48550/ARXIV.2501.12948. URL: https://arxiv.org/abs/2501.12948.

[10] Django. *django Web Framework*. last accessed: 2024-10-08. 2024. URL: https://www.djangoproject.com/.

[11] Liming Dong, Qinghua Lu, and Liming Zhu. *AgentOps: Enabling Observability of LLM Agents*. 2024. DOI: 10.48550/ARXIV.2411.05285. URL: https://arxiv.org/abs/2411.05285.

[12] Ximing Dong et al. *PromptExp: Multi-granularity Prompt Explanation of Large Language Models*. under review, preprint available on arXiv. 2024.

[13] Dynatrace. *Dynatrace*. last accessed: 2024-10-08. 2024. URL: https://www.dynatrace.com/.

[14] Daniel Fried et al. "InCoder: A Generative Model for Code Infilling and Synthesis". In: *Proceedings of the 11th International Conference on Learning Representations*. 2023.

[15] *GDB: The GNU Project Debugger*. https://sourceware.org/gdb/. Accessed: 2025-05-26.

[16] GitHub. *GitHub Copilot*. last accessed: 2024-10-09. 2024. URL: https://github.com/features/copilot.

[17] Siyuan Guo et al. "DS-Agent: Automated Data Science by Empowering Large Language Models with Case-Based Reasoning". In: *Proceedings of the 41st International Conference on Machine Learning*. 2024. URL: https://proceedings.mlr.press/v235/guo24b.html.

[18] *Haptik, Drive Business Efficiency at Scale with Generative AI*. last accessed: 2024-10-08. 2024. URL: https://www.haptik.ai/.

[19] Ahmed E. Hassan et al. "Rethinking Software Engineering in the Era of Foundation Models: A Curated Catalogue of Challenges in the Development of Trustworthy FMware". In: *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. 2024. ISBN: 9798400706585. DOI: 10.1145/3663529.3663849. URL: https://doi.org/10.1145/3663529.3663849.

[20] Ahmed E. Hassan et al. *Towards AI-Native Software Engineering (SE 3.0): A Vision and a Challenge Roadmap*. 2024. URL: https://arxiv.org/abs/2410.06107.

[21] Helicone. *Helicone*. last accessed: 2024-10-08. 2024. URL: https://www.helicone.ai/.

[22] Dan Hendrycks et al. "Measuring Massive Multitask Language Understanding". In: *International Conference on Learning Representations*. 2021.

[23] Shengran Hu, Cong Lu, and Jeff Clune. *Automated Design of Agentic Systems*. 2024. URL: https://arxiv.org/abs/2408.08435.

[24] Lei Huang et al. *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions*. 2023. URL: https://arxiv.org/abs/2311.05232.

[25] Humanloop. *Humanloop*. last accessed: 2024-10-08. 2024. URL: https://humanloop.com/.

[26] *Introducing Devin, the first AI software engineer*. last accessed: 2024-10-08. 2024. URL: https://www.cognition.ai/blog/introducing-devin.

[27] M. Izadi et al. "Language Models for Code Completion: A Practical Evaluation". In: *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 2024. DOI: 10.1145/3597503.3639138. URL: https://doi.ieeecomputersociety.org/10.1145/3597503.3639138.

[28] Carlos E. Jimenez et al. *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* 2024. URL: https://arxiv.org/abs/2310.06770.

[29] LangChain. *LangSmith by LangChain*. last accessed: 2024-10-08. 2024. URL: https://www.langchain.com/langsmith/.

[30] Langfuse. *Langfuse*. last accessed: 2024-10-08. 2024. URL: https://langfuse.com/.

[31] B. Li et al. "Enjoy your observability: an industrial survey of microservice tracing and analysis". In: *Empirical Software Engineering* 27.1 (2022). DOI: 10.1007/s10664-021-10063-9.

[32] Xiao-Hui Li et al. "A Survey of Data-Driven and Knowledge-Aware eXplainable AI". In: *IEEE Transac-*

tions on Knowledge and Data Engineering 34.1 (2022), pp. 29–49. DOI: 10.1109/TKDE.2020.2983930.

[33] Q. Vera Liao and Jennifer Wortman Vaughan. *AI Transparency in the Age of LLMs: A Human-Centered Research Roadmap*. 2023. arXiv: 2306.01941 [cs.HC]. URL: https://arxiv.org/abs/2306.01941.

[34] Ryan Liu et al. *Mind Your Step (by Step): Chain-of-Thought can Reduce Performance on Tasks where Thinking Makes Humans Worse*. 2024. arXiv: 2410.21333 [cs.LG]. URL: https://arxiv.org/abs/2410.21333.

[35] Lunary. *Lunary*. last accessed: 2024-10-08. 2024. URL: https://lunary.ai/.

[36] Tula Masterman et al. *The Landscape of Emerging AI Agent Architectures for Reasoning, Planning, and Tool Calling: A Survey*. 2024. URL: https://arxiv.org/abs/2404.11584.

[37] Nebuly. *Nebuly - Explicit and Implicit LLM User Feedback Quickguide*. last accessed: 2024-10-08. 2024. URL: https://www.nebuly.com/blog/explicit-implicit-llm-user-feedback-quickguide.

[38] OpenAI. *Introducing OpenAI o1*. last accessed: 2024-10-08. 2024. URL: https://openai.com/o1/.

[39] Qwak. *Qwak - LLMops*. last accessed: 2024-10-08. 2024. URL: https://www.qwak.com/products/llmops.

[40] Noah Shinn et al. "Reflexion: language agents with verbal reinforcement learning". In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. NIPS '23. New Orleans, LA, USA: Curran Associates Inc., 2023.

[41] Benjamin H. Sigelman et al. *Dapper, a Large-Scale Distributed Systems Tracing Infrastructure*. Tech. rep. Google, Inc., 2010. URL: http://research.google.com/archive/papers/dapper-2010-1.pdf.

[42] DiJia Su et al. *Dualformer: Controllable Fast and Slow Thinking by Learning with Randomized Reasoning Traces*. 2025. arXiv: 2410.09918 [cs.AI]. URL: https://arxiv.org/abs/2410.09918.

[43] Traceloop. *Traceloop*. last accessed: 2024-10-08. 2024. URL: https://www.traceloop.com/.

[44] Xingyao Wang et al. *OpenHands: An Open Platform for AI Software Developers as Generalist Agents*. 2025. arXiv: 2407.16741 [cs.SE]. URL: https://arxiv.org/abs/2407.16741.

[45] Xuezhi Wang et al. *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. 2023. arXiv: 2203.11171 [cs.CL]. URL: https://arxiv.org/abs/2203.11171.

[46] Jason Wei et al. "Chain-of-thought prompting elicits reasoning in large language models". In: *Proceedings of the 36th International Conference on Neural Information Processing Systems*. 2022. ISBN: 9781713871088.

[47] Weights and Biases. *Weights and Biases - Weave*. last accessed: 2024-10-08. 2024. URL: https://wandb.ai/site/weave/.

[48] WhyLabs. *WhyLabs*. last accessed: 2024-10-08. 2024. URL: https://whylabs.ai/.

[49] Fangzhi Xu et al. *Are Large Language Models Really Good Logical Reasoners? A Comprehensive Evaluation and Beyond*. 2024. URL: https://arxiv.org/abs/2306.09841.

[50] Shunyu Yao et al. "Tree of thoughts: deliberate problem solving with large language models". In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. 2024.

[51] Yuntong Zhang et al. "AutoCodeRover: Autonomous Program Improvement". In: *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2024. ISBN: 9798400706127. DOI: 10.1145/3650212.3680384. URL: https://doi.org/10.1145/3650212.3680384.