VeriLA: A Human-Centered Evaluation Framework for Interpretable Verification of LLM Agent Failures

Yoo Yeon Sung†
University of Maryland
USA
yysung53@umd.edu

Hannah Kim
Megagon Labs
USA
hannah@megagon.ai

Dan Zhang
Megagon Labs
USA
dan_z@megagon.ai

Abstract

AI practitioners increasingly use large language model (LLM) agents in compound AI systems to solve complex reasoning tasks, these agent executions often fail to meet human standards, leading to errors that compromise the system's overall performance. Addressing these failures through human intervention is challenging due to the agents' opaque reasoning processes, misalignment with human expectations, the complexity of agent dependencies, and the high cost of manual inspection. This paper thus introduces a human-centered evaluation framework for **V**erifying **LL**M **A**gent failures (VeriLA), which systematically assesses agent failures to reduce human effort and make these agent failures interpretable to humans. The framework first defines clear expectations of each agent by curating human-designed agent criteria. Then, it develops a human-aligned agent verifier module, trained with human gold standards, to assess each agent's execution output. This approach enables granular evaluation of each agent's performance by revealing failures from a human standard, offering clear guidelines for revision, and reducing human cognitive load. Our case study results show that VeriLA is both interpretable and efficient in helping practitioners interact more effectively with the system. By upholding accountability in human-agent collaboration, VeriLA paves the way for more trustworthy and human-aligned compound AI systems.

CCS Concepts

• **Computing methodologies** → **Machine learning**; **Natural language processing**; • **Human-centered computing** → *Human computer interaction (HCI).*

Keywords

LLM-as-agent, compound AI system, human-centered agent evaluation, interpretable agents, human-agent interaction

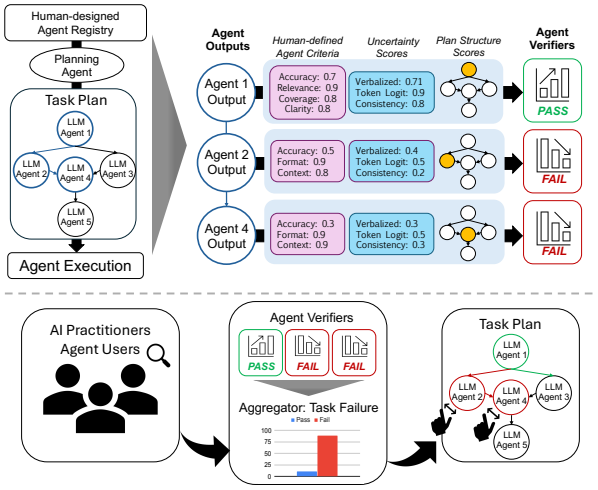

Figure 1: Overview of VeriLA. Our framework operates in three main stages (1) planning where a planning agent decomposes a task into subtasks using a human-designed agent registry and generates a plan graph; (2) agent execution where specialized LLM agents perform the subtasks; and (3) execution verification, which verifies each LLM agent's outputs based on human-defined agent criteria, agent uncertainty, and dependency information from the plan structure. We then assess task failure with aggregation metrics that combine verifier scores. Our framework guides users to detect task failures efficiently, identify faulty agents, and analyze the root causes of their failure.

1 Introduction

As large language models (LLMs) continue to excel across various fields, they are increasingly used to address complex reasoning tasks through LLM-as-agent systems [35, 38]. A key application is a LLM-based compound AI system, where a planning agent breaks a complex task into simpler subtasks, and delegates these subtasks to multiple specialized LLM agents [37, 41, 42]. Each assigned agent must accurately execute its subtask, as the final agent's output heavily relies on the previous agents' outputs and is considered as the final solution of the overall task [6]. Failures in any agent can propagate and cause the overall task failure [17, 32]. While these compound systems demonstrate strong problem-solving capabilities, they face significant limitations in that they may produce outputs that contradict human expectations, often requiring human intervention for failure feedback and revision [2, 20]. However, providing feedback is challenging because agent outputs come from reasoning that deviates from humans or lack clarity on the cause of failure, hindering users from providing guidance on remedying execution failures. Moreover, manually reviewing each step is labor intensive and not scalable. It forces users to audit each agent's execution outputs, increasing the risk of errors. This underscores the need for more systematic and

† Work done during an internship at Megagon Labs.

efficient methods for auditing and supervising compound AI systems with LLM agents.

To address this challenge, our framework consists of three interconnected components: (1) **planning**, where a planning agent decomposes the target task into simpler subtasks based on a predefined agent registry, generating a graph-based plan. The agent registry provides each agent with clear role assignments and execution guidelines that adhere to human expectations; (2) **agent execution**, in which LLM agents sequentially perform their designated subtasks, with success determined by their adherence to the predefined roles; (3) **execution verification**, where a verifier module automatically assesses each agent output to ensure its success in fulfilling its assigned subtask. Our verifier incorporates human-centered judgments on agent outputs along with its relationship with other agents: an agent's subtask type, scores based on human-defined agent criteria, agent uncertainty, and agent's dependency within the plan structure. In addition to verifying the execution of individual agents, VeriLA provides additional verification of overall task success. We introduce a metric that aggregates verifier results across agents to identify failed tasks due to agent failures. By leveraging the plan's graph structure, it enables targeted analysis of verifiers and failure criteria, streamlining failure detection. To evaluate our framework, we conducted a case study on a complex reasoning task—solving mathematical reasoning problems—demonstrating its effectiveness and practical applicability to AI practitioners.

In summary, VeriLA enables detailed auditing of agentic systems to ensure transparency in agent failures, reducing manual review efforts, and strengthening trust between users and compound AI systems. Beyond merely automating the validation of each agent's success, we hope for a collaborative problem-solving between human and LLM agents by tailoring agent workflow to human needs.

2 Related Work

2.1 Need for Human Intervention in Compound AI Systems

LLMs are capable of reasoning, tool usage, planning, and following instructions [24]. For tasks that are too complex for n-shot CoT prompting, LLMs can decompose them into subtasks and delegate each subtask to several agents [14]. This decomposition externalizes LLMs' inner reasoning, allowing users to intervene at intermediate steps—such as suggesting alternatives or correcting errors—to ensure alignment with human expertise and domain-specific requirements [2, 20, 23, 37]. While recent LLMs can plan and self-refine tasks [14], issues such as hallucinations and limited feedback scopes remain [33], making human intervention crucial to detect and correct errors before they propagate [13]. For example, errors such as misinterpreting a calculation or producing an inaccurate response may go unnoticed by LLM-only evaluation but can be handled by humans. Thus, while LLMs can help with task decomposition and execution [24], the integration of human intervention remains essential for reliability and accuracy, especially in error-sensitive domains such as legal or medical applications.

2.2 Human-Centered Evaluation of Language Models

Narrowing the socio-technical gap is a key challenge in HCI, focusing on how evaluation results should be utilized and by whom. Liao and Xiao [22] argue that evaluation modules must assess how well human needs are met in downstream use cases. Similarly, Arabzadeh et al. [1] emphasize the importance of identifying criteria for LLMs. For example, in math reasoning tasks, an agent's success depends not only on producing correct solutions but on presenting them with completeness, conciseness, and clarity—critical criteria for such task [16]. Also, user-defined criteria improve alignment between LLM evaluator outputs and human judgements [19]. Shankar et al. [30] propose an approach to align LLM evaluators with human-defined criteria, tailoring evaluations to specific use cases. Building on this line of work, we adopt human-defined agent criteria to evaluate agent's execution success in our framework.

2.3 Assessing LLM Agent's Uncertainty

Several techniques have been proposed to assess the uncertainty level of LLMs, including methods based on token likelihoods, consistency across multiple prompts, and self reflection [5]. A common approach leverages token likelihood, such as logit-based confidence and entropy-based confidence, which rely on the LLM's internal probability distribution over potential tokens [15] (all uncertainty methods detailed in Appendix B.1). Another approach, verbalized confidence [39], asks the LLM to articulate its confidence, either as qualitative indicators or explicit confidence scores. Uncertainty can also be assessed using external LLM evaluators, utilizing verbalized confidence from their response or logit-based confidence from its verification assessment (e.g., logits for the prediction of a "TRUE" token) [10]. Additionally, self-consistency methods [36] compare outputs from multiple runs: frequency-based metrics [40] assess agreement across outputs, while others use weighted aggregation with logit or verbalized confidence [39].

3 VeriLA: Framework for Verifying LLM Agents in Compound AI Systems

We propose an evaluation framework that aims to assist users in auditing and interacting with compound AI systems. During users' inspection of overall task failures, they can detect each agent's execution failures, and quickly and clearly understand the reasons behind the failure. This enables them to provide actionable suggestions for planning or execution revisions.

3.1 Planning

In a compound AI system, a planning agent's role is to decompose a task into a sequence of subtasks, assign them to

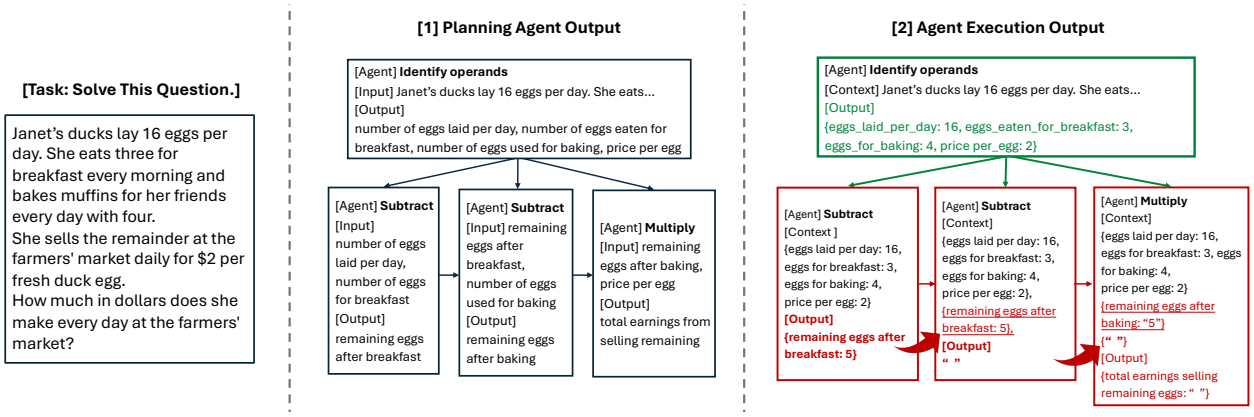

Figure 2: Example of agent's failure propagating to overall task failure. For example, based on the generated plan from the planning agent, each agent should accurately execute their subtasks. The first "subtract" agent failed to calculate the remaining eggs, causing subsequent "subtract" and "multiply" agents to lack the necessary context for a successful execution (three red boxes). An agent-specific verifier can help users trace the error propagation, identify the root cause of the error, and understand how it led to the task failure.

specialized agents—which are typically predefined in the system's agent registry—and generate a plan to solve the task.[1]

To curate an agent registry that aligns with the human reasoning process, we first ask AI practitioners to curate a system's agent registry tailored to the target application. They register each agent with a specific role, along with its expected input and output. For example, in a math reasoning task, an "Add" agent is responsible for summing given operands, where the input is a list of numbers, and the output is a single sum (Table 1). [2] Using this agent registry, the planning agent decomposes a given task into subtasks and delegates them to appropriate agents (Upper left in Figure 1). This later helps users in diagnosing each agent's failure; it informs them whether the agent was executed appropriately based on its role and has a proper output format.

Then, our planning agent generates a plan with a directed acyclic graph (DAG) format, where each node represents an agent, and directed edges show input-output dependencies between the nodes (example in Appendix A). This graph-based planning ensures that the task complexity is decomposed into an interrelated sequence of simplified subtasks. Then, each subtask is assigned to an appropriate agent with specific inputs and outputs, mirroring how humans often approach complex tasks [11].

Planning agents also generate instruction prompts for each agent, reflecting its role and input–output format. These prompts should be closely tied to the original task, reducing the risk of agents hallucinating or forgetting the trajectory needed to solve the task.

[1]The planning agent is not registered in agent registry and thus does not participate in the plan.

[2]To guide practitioners on the necessary agents and their required functionalities, we use Chain-of-Thought (CoT) prompting to generate a pool of agent candidates. Then, they identify the most common agents and craft their roles, inputs, and outputs (Appendix D).

3.2 Agent Execution

Agents execute assigned specific subtasks and instruction prompts from the generated plan. Additionally, they receive relevant context information as input which includes relevant outputs produced from the preceding agents. Because agents can fail during execution, users must intervene to correct errors and prevent an agent's failure from propagating to overall task failure. However, manually auditing whether an agent has fulfilled its subtask is both cumbersome and cognitively demanding for users. Thus, we introduce a agent-specific verifier in the next section.

3.3 Execution Verification by Human-Aligned Agent Verifier

Our agent verifier autonomously evaluates agent executions and flags potentially incorrect results, enabling users to focus only on the problematic agents within the plan. Although some self-verifying agents exist [25, 33], we question their reliability [31], lack of contextual understanding [27], and insufficient human alignment [12].

To address these issues, we build a separate evaluation module for each agent to detect execution failures. This module functions as a binary classifier, trained on multiple features such as human-defined agent criteria, agent uncertainty, and plan structures with subtask types. To balance verifier's autonomy with human-designed criteria, we integrate external LLM judge scores that are assessed with human-defined criteria (Appendix E). These criteria are carefully tailored for each agent to ensure that expectations align with human standards. This way, our verifier ensures that each agent aligns with human expectations and is evaluated according to users' specific needs [22]. For instance, the "Subtract" agent's execution is evaluated not only on the accuracy of summation but also on its adherence to the expected format (e.g., number) and

sufficiency of context information for the subtask (Figure 2). Grounding agents' evaluation in predefined criteria permits objective judgment and clarifies failure reasons.

Additionally, we integrate three LLM uncertainty estimation techniques: verbalized confidence, logit-based confidence, and confidence based on self-consistency. This is under the assumption that lower certainty often correlates with a higher likelihood of errors or deviations (details on uncertainty features in Appendix B.1). Finally, we include each agent's subtask type and its position within the plan's DAG structure, by using subtask categorical encoding and features like the number of preceding nodes to capture dependency relationships between agents (Appendix B.3). These structural features enhance our verifier's ability to assess execution reliability within the overall task.

Ground Truth to Train Agent Verifier. We use human annotated labels as ground truth to train our agent verifier. By learning execution feature patterns from these labels, the model predicts whether an agent is likely to fail during execution based on human expectations. To ensure accurate and fair labeling, human annotators assess if the agent correctly performed its task, following the agent registry roles and human-defined criteria. Annotators are given context information, clear role definitions, expected output format, and the same human-defined criteria used by LLM judges.

3.4 Aggregation Metrics for Overall Task Failure Prediction

To support human-agent interaction, we guide users in identifying task failures when decomposed and executed by agents. Our verifier predicts potential failures and provides confidence scores for each agent's execution, which we use to assess overall task success. We propose several aggregation metrics to aggregate individual verifier scores. The metric scores represent the likelihood of overall task success, with higher scores indicating a greater chance of success.

We begin with simple methods such as selecting the lowest score among all subtasks (*min* aggregator), highlighting the weakest agent execution; and computing the arithmetic average of all scores (*mean* aggregator), providing a balanced view of subtask performances.

We consider the relative importance of agents within the plan structure, as overall task performance depends on how effectively an agent's output transfers to others. To capture this, we propose two structural metrics for weighting agent scores. The distance-based metric emphasizes agents positioned closer to the source or sink nodes in the plan graph, under the assumption that these agents play a more critical role in the execution flow. This metric weights each agent's score inversely proportional to its distance from either the source or sink node (*source distance* or *sink distance* aggregator), ensuring that agents with greater structural influence have a higher impact on the aggregated score. Given the overall task T consisting of subtasks fulfilled by agents $\{S_1, S_2, \ldots, S_m\}$, we denote the i-th agent S_i's verifier score as $\hat{y}_i$, which is predicted based on features extracted from its execution outputs.

$$\text{AggScore}_{\text{dist}}\left(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}m\right) = \frac{\sum i = 1^m \frac{\hat{y}i}{d_i}}{\sum i = 1^m \frac{1}{d_i}}, \quad (1)$$

where d_i denotes the shortest path distance from agent S_i.

Similarly, the degree-based metric assigns higher weights to agents with greater connectivity, reflecting their influence within the overall plan structure. It weights each agent's score based on the indegree or outdegree (*indegree* or *outdegree* aggregator) of its node:

$$\text{AggScore}_{\text{deg}}\left(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}m\right) = \frac{\sum i = 1^m \text{deg}_i \cdot \hat{y}i}{\sum i = 1^m \text{deg}_i}, \quad (2)$$

where $\hat{y}_i$ represents the prediction score for subtask S_i, and deg_i denotes the degree (either indegree or outdegree) of subtask S_i in the plan DAG.

We later assess the accuracy of the aggregated verification results by comparing the task's gold label, indicating actual plan success, with the final agent label predicted by the verifier. This comparison evaluates how effectively the verifier is aggregated to predict task failure (§ 4.3).

4 Case Study: Mathematical Reasoning

We demonstrate the effectiveness of VeriLA on mathematical reasoning tasks. Math reasoning problems can be naturally decomposed into step-by-step plans, allowing us to focus on evaluating agent execution failures while ensuring that the results are easily verifiable by humans.

4.1 Experiment Setting

4.1.1 Datasets. We evaluate our pipeline on four math reasoning datasets: GSM8K [7] and Date Understanding (C3), Multi-Step Arithmetic (C11), Object Counting (C13) from BIG-Bench Hard (BBH) [34]. The GSM8K dataset consists of grade-school-level math word problems written in natural language, whereas the Multi-Step Arithmetic dataset is presented in equation format and involves more complex problems that require multiple steps. Date Understanding focuses on reasoning about and performing operations with dates, while Object Counting involves enumerating objects of interests.

4.1.2 Planning and Agent Execution. For each task, our planning agent generates a DAG plan to solve it using a human-designed agent registry (detailed in Table 1). We manually filter out instances where the generated plans are invalid,[3] keeping only those with valid plans. The average number of subtasks per plan is 4 for GSM8K, 3.5 for Date Understanding, 2.5 for Object Counting, and 5.6 for Multistep Arithmetic. Each subtask in a plan is then executed by an assigned agent. Both the planning agent and agents in agent registry use GPT-4o

[3]We consider both structural validity (e.g., missing dependencies between subtasks) and semantic correctness (e.g., incorrect agent assignment or faulty instructions).

Table 1: Human-designed agent registry for mathematical reasoning tasks.

Agent	Role	Input	Output	Output Format
Identify Operands	Identify operands with text description of each operands	Math question	List of operand names with their values	{<name>: Number, ...}
Add	Add numbers or dates	List of operands	One summed value	Number or Date
Subtract	Subtract numbers or dates	List of operands	One subtracted value	Number or Date
Multiply	Multiply numbers	List of operands	One multiplied value	Number
Divide	Divide numbers	List of operands	One divided value	Number
Filter	Filter a list based on a condition	List, condition	Filtered list	List
Sort	Sort a list by an attribute	List, attribute	Sorted list	List
Convert Format	Convert input from one format to another format	Text, format	Formatted text	Text
Date Lookup	Identify year, month, and day from a natural language description	Text	Date	Date

with a temperature of 0.1. All prompts for planning, execution, and verification are provided in Appendix C.

4.1.3 Agent Execution Verfication.

Gold (Execution Failure) Label Annotation. To train the verifier, human annotators label agent's execution failures based on agent criteria, expected inputs and outputs, and input information (§ 3.3). For GSM8K, we collected 1,975 subtasks from 497 tasks, each subtask labeled by three annotators from crowdsourcing (Appendix F). Due to low inter-rater reliability (Fleiss' kappa: 0.41), only unanimously labeled samples (973 subtasks) were retained for training. For the BBH datasets, the three authors handled annotations to ensure quality while reducing costs.

Feature Collection. We extracted 26 execution features as described in § 3.3. For self-consistency, we used the same model as the corresponding execution agent with a temperature of 0.7 to generate diverse outputs, following Xiong et al. [39]. For external LLM judges, we employed GPT-4o with a temperature of 0.1.

4.2 Verifier Results for Agent Failures

Using features from all agent outputs in the plan, we train a simple machine learning model, Random Forest model, which achieve a high average accuracy (0.88%) across four datasets.[4], suggesting the verifier's effectiveness in identifying failed agent executions.

Next, we investigate whether its performance varies across different agents. As shown in Figure 3, the test accuracy remains consistently high across various subtasks except "identify operands" in GSM8K where agents often struggle with accurate formatting. Moreover, similar subtasks—such as "Add" and "Subtract", which share the same subjective criteria—exhibit comparable test accuracies across all datasets, suggesting their generalizability to other tasks.

To further analyze verifier behavior, we conduct an ablation study with different features to predict agent execution failures. Our verifier achieves the highest performance with all features included while excluding any single feature leads to a drop in performance (Figure 4). This suggests that the features provide complementary information, each playing a distinct role in model accuracy. Notably, the agent criteria features has the largest performance drop when removed, suggesting that incorporating human-defined agent criteria from external LLM judges improves the verifiers' ability to align their predictions with human priorities. On the other hand, the consistency-related features had minimal impact on performance, implying they may be less critical.

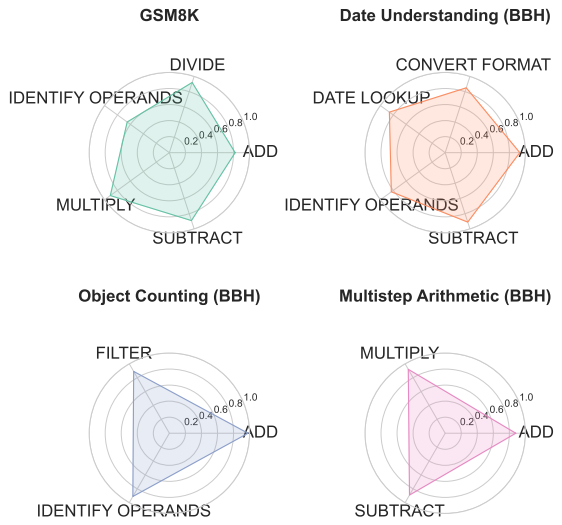

Figure 3: Verifier accuracy across datasets. The test accuracy remains consistently high across subtasks, without bias toward any specific one. Similar subtasks, like "Add" and "Subtract," which share the same criteria, also show comparable accuracies across all datasets.

4.3 Aggregator Results for Overall Task Failures

To predict whether the overall task is failing, we present a few aggregation metrics (aggregator), which combine subtask

[4]We compared several ML models and select the one with the highest accuracy (Appendix G.2).

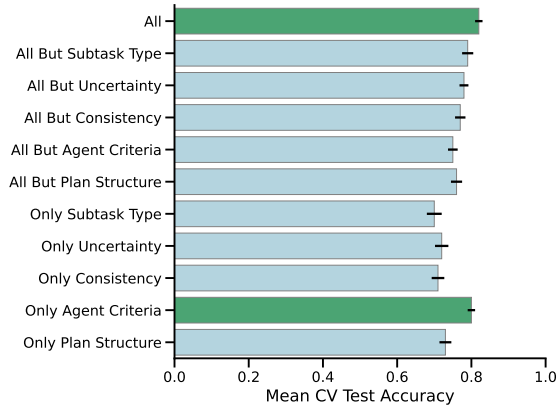

Figure 4: Ablation study on different feature configurations evaluating verifiers' test accuracy. Human-defined agent criteria feature enhances its performance, showing the highest accuracy when all features are used.

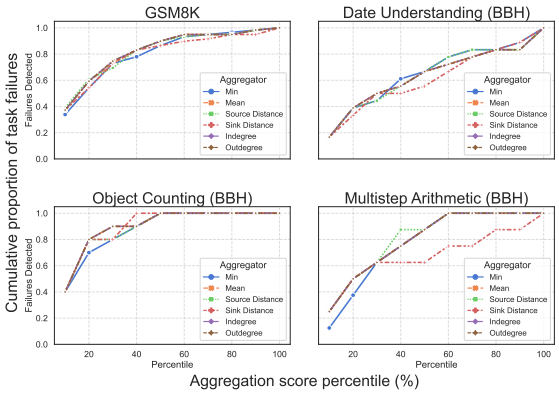

Figure 5: Aggregation performance measured by failure rate across aggregation score percentiles. They all show an upward trend, suggesting that they can help users prioritize tasks more likely to fail, when the labor budget is limited, allowing auditing of high-risk tasks first. Overall, *mean* and *outdegree* showed stable performance across datasets and can be used as default aggregation metrics for new datasets.

verification scores, that function as a *task-level verifier*.[5] By leveraging the aggregated score from the task verifier, users can prioritize and investigate tasks that are most likely to generate false LLM outputs. They can then analyze the reasons behind a task's failure by examining the flagged subtasks identified by the agent verifier.

Figure 5 shows task-level verification performance from different aggregation methods, with lower score indicating a greater chance of failure. x-axis represents percentiles of ranked aggregation scores, and y-axis shows the cumulative ratio of detected failures within each percentile relative to all failed tasks. The curves closer to the top-left corner indicate better performance. *Sink distance* aggregator identified all failures fastest in object counting but was slowest in date understanding. *Source distance* aggregator generally outperformed *sink distance*, except in GSM8K. This suggests that, for GSM8K, proximity to the starting nodes is a stronger indicator than proximity to the final node; GSM8K's typical first subtask—identifying operands—is a common source of error. Although no single aggregator consistently outperforms others across all datasets, they all show an upward trend, suggesting that they can help users prioritize tasks more likely to fail. This can be especially useful when labor is limited, allowing auditing of high-risk tasks first. Overall, *mean* and *outdegree* showed stable performance across datasets and can be used as default aggregation metrics for new datasets.

5 Conclusion and Future Work

In this work, we introduce VeriLA, a human-centered evaluation framework that verifies agent execution failures; this encourages reliability and interpretability in humans using compound AI systems. By applying a verifier that assesses each agent's outputs through a combination of human-defined criteria, agent output uncertainty, and agent dependencies within the plan. Thus, VeriLA also captures error propagation within the plan, facilitating better human-agent interaction. We also present a verifier-driven task failure metric that help users detect tasks prior to their auditing. Thus, VeriLA enhances accountability on agent performance and labor efficiency by enabling granular human inspection of failing agents and the underlying reasons for their failures.

Future work remains for open agentic systems that involve diverse execution agents with more complexity and dynamic error propagation, highlighting the need for robust, human-centered evaluation methods for human-agent interaction. For future work, we plan to conduct a crowdsourced user study that evaluates VeriLA's subtask and task-level verification performance and usability, comparing outcomes with and without VeriLA. Second, building on our aggregation metrics, we will develop an advanced aggregator that incorporates user-centric signals like free-form user feedback and real-time incentives. Finally, we will expand our framework to broader applications, such as open-domain question answering and fact-checking, to support a wider range of human-agent interactive systems (Appendix D.1). To support new domains, we will automate agent curation by extracting subtasks via Chain-of-Thought decomposition and clustering them to recommend suitable agents. To conclude, we aim to cultivate synergy between humans and LLM agents by designing compound AI systems that align with human needs, enabling collaboration on real-world tasks that require intricate problem-solving.

[5] A task is considered successfully solved if the last step's execution output is linguistically equivalent [21] to the gold answer from the original dataset.

References

[1] Negar Arabzadeh, Siqing Huo, Nikhil Mehta, Qingyun Wu, Chi Wang, Ahmed Hassan Awadallah, Charles L. A. Clarke, and Julia Kiseleva. 2024. Assessing and Verifying Task Utility in LLM-Powered Applications. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 21868–21888. doi:10.18653/v1/2024.emnlp-main.1219

[2] Ian Arawjo, Chelse Swoopes, Priyan Vaithilingam, Martin Wattenberg, and Elena L. Glassman. 2024. ChainForge: A Visual Toolkit for Prompt Engineering and LLM Hypothesis Testing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 304, 18 pages. doi:10.1145/3613904.3642016

[3] Léon Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT 2010)*. Physica-Verlag HD, 177–186.

[4] Leo Breiman, Jerome Friedman, Richard A Olshen, and Charles J Stone. 1986. *Classification and Regression Trees*. Wadsworth.

[5] Jiuhai Chen and Jonas Mueller. 2024. Quantifying uncertainty in answers from any language model and enhancing their trustworthiness. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 5186–5200.

[6] Yuheng Cheng, Ceyao Zhang, Zhengwen Zhang, Xiangrui Meng, Sirui Hong, Wenhao Li, Zihao Wang, Zekai Wang, Feng Yin, Junhua Zhao, et al. 2024. Exploring Large Language Model based Intelligent Agents: Definitions, Methods, and Prospects. *https://arxiv.org/abs/2401.03428* (2024).

[7] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *https://arxiv.org/pdf/2110.14168* (2021).

[8] David R Cox. 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)* 20, 2 (1958), 215–242.

[9] Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. System Sci.* 55, 1 (1997), 119–139.

[10] Jiahui Geng, Fengyu Cai, Yuxia Wang, Heinz Koeppl, Preslav Nakov, and Iryna Gurevych. 2024. A Survey of Confidence Estimation and Calibration in Large Language Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 6577–6595. doi:10.18653/v1/2024.naacl-long.366

[11] Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated Planning: theory and practice*. Elsevier-Morgan Kauffman.

[12] Nitesh Goyal, Minsuk Chang, and Michael Terry. 2024. Designing for Human-Agent Alignment: Understanding what humans want from their agents. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–6.

[13] Madeleine Grunde-McLaughlin, Michelle S. Lam, Ranjay Krishna, Daniel Weld, and Jeffrey Heer. 2025. Designing LLM Chains by Adapting Techniques from Crowdsourcing Workflows. https://doi.org/10.1145/3716134

[14] Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of LLM agents: A survey. *arXiv preprint arXiv:2402.02716* (2024).

[15] Yuheng Huang, Jiayang Song, Zhijie Wang, Shengming Zhao, Huaming Chen, Felix Juefei-Xu, and Lei Ma. 2023. Look Before You Leap: An Exploratory Study of Uncertainty Measurement for Large Language Models. arXiv:2307.10236 [cs.SE]

[16] Lujain Ibrahim, Saffron Huang, Lama Ahmad, and Markus Anderljung. 2024. Beyond static AI evaluations: advancing human interaction evaluations for LLM harms and risks. *arXiv preprint arXiv:2405.10632* (2024).

[17] Laure Jaeger, Tom Jorquera, Sylvain Lemouzy, Christian Gogu, Stéphane Segonds, and Christian Bes. 2013. Uncertainty propagation in multi-agent systems for multidisciplinary optimization problems. In *10th World Congress on Structural and Multidisciplinary Optimization (WCSMO 10)*. pp–1.

[18] Geoffrey H John and Pat Langley. 1995. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI 1995)*. Morgan Kaufmann, 338–345.

[19] Tae Soo Kim, Yoonjoo Lee, Jamin Shin, Young-Ho Kim, and Juho Kim. 2024. EvalLM: Interactive Evaluation of Large Language Model Prompts on User-Defined Criteria. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 306, 21 pages. doi:10.1145/3613904.3642216

[20] LangChain. 2013. LangChain. https://github.com/langchain-ai/langchain.

[21] Zongxia Li, Ishani Mondal, Huy Nghiem, Yijun Liang, and Jordan Lee Boyd-Graber. 2024. PEDANTS: Cheap but Effective and Interpretable Answer Equivalence. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, Miami, Florida, USA, 9373–9398. doi:10.18653/v1/2024.findings-emnlp.548

[22] Q Vera Liao and Ziang Xiao. 2023. Rethinking model evaluation as narrowing the socio-technical gap. *arXiv preprint arXiv:2306.03100* (2023).

[23] David Chuan-En Lin and Nikolas Martelaro. 2024. Jigsaw: Supporting Designers to Prototype Multimodal Applications by Chaining AI Foundation Models. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 4, 15 pages. doi:10.1145/3613904.3641920

[24] Renze Lou, Kai Zhang, and Wenpeng Yin. 2024. Large Language Model Instruction Following: A Survey of Progresses and Challenges. *Computational Linguistics* 50, 3, 1053–1095.

[25] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems* 36 (2024).

[26] Aakash Parmar, Rakesh Katariya, and Vatsal Patel. 2019. A review on random forest: An ensemble classifier. In *International conference on intelligent data communication technologies and internet of things (ICICI) 2018*. Springer, 758–763.

[27] Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2024. ADaPT: As-Needed Decomposition and Planning with Language Models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 4226–4252. doi:10.18653/v1/2024.findings-naacl.264

[28] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *Nature* 323, 6088 (1986), 533–536.

[29] Robert E. Schapire. 2013. *Explaining AdaBoost*. Springer Berlin Heidelberg, Berlin, Heidelberg. 37–52 pages. doi:10.1007/978-3-642-41136-6_5

[30] Shreya Shankar, JD Zamfirescu-Pereira, Björn Hartmann, Aditya Parameswaran, and Ian Arawjo. 2024. Who validates the validators? aligning llm-assisted evaluation of llm outputs with human preferences. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–14.

[31] Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. 2024. On the Self-Verification Limitations of Large Language Models on Reasoning and Planning Tasks. *arXiv e-prints* (2024), arXiv–2402.

[32] Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. 2023. Cognitive architectures for language agents. *Transactions on Machine Learning Research* (2023).

[33] Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. AdaPlanner: Adaptive Planning from Feedback with Language Models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

[34] Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 13003–13051. doi:10.18653/v1/2023.findings-acl.824

[35] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science* 18, 6 (2024), 186345.

[36] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H Chi, Quoc V Le, and Denny Zhou. 2022. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *International Conference on Learning Representations (ICLR)*.

[37] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 385, 22 pages. doi:10.1145/3491102.3517582

[38] Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2025. The rise

and potential of large language model based agents: A survey. *Science China Information Sciences* 68, 2 (2025), 121101.

[39] Miao Xiong, Zhiyuan Hu, Xinyang Lu, YIFEI LI, Jie Fu, Junxian He, and Bryan Hooi. 2024. Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs. In *The Twelfth International Conference on Learning Representations*.

[40] Gal Yona, Roee Aharoni, and Mor Geva. 2024. Can Large Language Models Faithfully Express Their Intrinsic Uncertainty in Words?. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, 7752–7764. doi:10.18653/v1/2024.emnlp-main.443

[41] Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. 2024. The Shift from Models to Compound AI Systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/.

[42] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua Tenenbaum, Tianmin Shu, and Chuang Gan. [n. d.]. Building Cooperative Embodied Agents Modularly with Large Language Models. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*.

A An Example Plan from Planning Agent


```
{
    "id_": 0,
    "question": "Janet's ducks lay 16 eggs per day.
    She eats three for breakfast every morning and
    bakes muffins for her friends every day with four.
    She sells the remainder at the farmers' market daily
    for $2 per fresh duck egg. How much in dollars does
    she make every day at the farmers' market?",
    "answer": "Janet sells 16 - 3 - 4 = <<16-3-4=9>>9
    duck eggs a day.\nShe makes 9 * 2 = $<<9*2=18>>18
    every day at the farmers' market.\n#### 18",
    "system_prompt": "You are a helpful assistant in
    solving math questions",
    "user_prompt": {
        "1": "Identify the number of eggs laid per day,
        eggs eaten for breakfast, eggs used for
        baking, and the price per egg from the question.",
        "2": "Subtract the number of eggs eaten for
        breakfast from the number of eggs laid per day.",
        "3": "Subtract the number of eggs used for
        baking from the remaining eggs after breakfast.",
        "4": "Multiply the number of eggs available for
        sale by the price per egg."
    },
    "plan": [
        {
            "id": 1,
            "name": "identify operands",
            "input": "question",
            "output": "number of eggs laid per day,
            eggs eaten for breakfast, eggs used for
            baking, price per egg"
        },
        {
            "id": 2,
            "name": "subtract",
            "input": "number of eggs laid per day,
            eggs eaten for breakfast",
            "output": "remaining eggs after breakfast"
        },
        {
            "id": 3,
            "name": "subtract",
            "input": "remaining eggs after breakfast,
                eggs used for baking",
            "output": "eggs available for sale"
        },
        {
            "id": 4,
            "name": "multiply",
            "input": "eggs available for sale,
            price per egg",
            "output": "total earnings from selling eggs"
        }
    ],
    "edges": [
        [1, 2],
        [2, 3],
        [1, 3],
        [3, 4],
        [1, 4]
    ]
}
```


B Agent Verifier Features

We collect features such as uncertainty metrics from LLM agents' internals during execution, consistency-related metrics from additional LLM prompting, criteria-based scores from external LLM judges, and node characteristics features. We use GPT-4o for all experiments, employing a temperature setting

of 0.7 for consistency-related metrics to obtain as diverse answers as possible, and a temperature of 0.1 for the remaining experiments [39].

B.1 Features on Agent Uncertainty

- **Verbalized confidence** [39] reflects how confident the executor module is about its output, often derived from explicit confidence scores or qualitative indications of certainty (e.g., "0.7").
- **Logit-based confidence** [15] reflects the average of the exponentials of the token log probabilities (LP):

$$LP_{avg} = \frac{1}{N}\sum_{i=1}^{N} p(s_j \mid x_i) = \frac{1}{N}\sum_{i=1}^{N} \exp(\log p(s_j \mid x_i)), \quad (3)$$

  where N is the number of tokens and $\log p(s_j \mid x) = \sum_{i=1}^{j} \log p(s_i \mid s_{<i})$, where s_i is the i-th output token and $s_{<i}$ denotes the set of previous tokens. We denote $\log p(s_j \mid x_i)$ as token log probability.
- **Softmax-based confidence** reflects the average of softmax values across the generated tokens, providing a measure of the overall uncertainty of the model based on the top-k token log probabilities:

$$Softmax_{avg} = \frac{1}{N}\sum_{n=1}^{N} \frac{\exp(\mathbf{z}_n)}{\sum_{i=1}^{k}\exp(\mathbf{z}_{n,i})}, \quad (4)$$

  where $\mathbf{z}_n$ is the token log probability and N is the number of tokens.
- **Entropy-based confidence** [15] reflects the average of entropy values across the generated tokens, providing a measure of the overall uncertainty of the model based on the top-k token log probabilities:

$$Entropy_{avg} = \frac{1}{N}\sum_{n=1}^{N}\left(-\sum_{i=1}^{k}\frac{\exp(\mathbf{z}_{n,i})}{\sum_{j=1}^{k}\exp(\mathbf{z}_{n,j})}\log\left(\frac{\exp(\mathbf{z}_{n,i})}{\sum_{j=1}^{k}\exp(\mathbf{z}_{n,j})}\right)\right), \quad (5)$$

  where $\mathbf{z}_n$ is the token log probability and $\mathbf{z}_{n,i}$ is the i-th top log probability.
- Features from the a separate LLM evaluator capture the uncertainty in its assessment of the initial LLM execution.
  - **Verbalized confidence from external LLM evaluator** is attained directly from the external evaluator's generated response.
  - **Logit-based confidence from LLM evaluator** is attained by the exponential of the logit value of the LLM evaluatorś verification assessment (e.g., logit value of the `TRUE`).
- Features using self-consistency [36] technique; we run the same prompt five times and aggregate the confidence values in the following ways:
  - **Self-consistency (Type A): frequency** [40] measures the confidence of the executor module by the degree of agreement among the candidate outputs and integrates the inherent uncertainty in the model's output [39]:

$$\text{Confidence}_{freq} = \frac{1}{M}\sum_{i=1}^{M} \mathbb{1}_{\{\text{agreement}(\hat{Y}_i,\tilde{Y})>\theta\}}, \quad (6)$$

    where $\mathbb{1}_{\{\text{condition}\}}$ is the indicator function that returns 1 if the candidate answer $\hat{Y}_i$ is consistent with the initial execution result $\tilde{Y}$ based on the agreement threshold θ, and 0 otherwise. Here, M denotes the number of candidate answers, and θ is the threshold for agreement. We use the answer equivalence package PEDANT [21] with their recommended threshold of 0.5 to assess the agreement.
  - **Self-consistency (Type B): verbalized confidence** [39] measures the average verbalized confidence among the subset of candidate answers identified as correct.

$$\text{Confidence}_{verb} = \frac{\sum_{i=1}^{M} C_i^{verb}\cdot \mathbb{1}_{\{\text{correctness}_i=1\}}}{\sum_{i=1}^{M}\mathbb{1}_{\{\text{correctness}_i=1\}}}, \quad (7)$$

    where M is the total number of candidate answers and C_i denotes the verbalized confidence of candidate answers.
  - **Self-consistency (Type C): logit-based** measures the average logit-based confidence among the subset of candidate answers identified as correct.

$$\text{Confidence}_{log} = \frac{\sum_{i=1}^{M} C_i^{log}\cdot \mathbb{1}_{\{\text{correctness}_i=1\}}}{\sum_{i=1}^{M}\mathbb{1}_{\{\text{correctness}_i=1\}}}, \quad (8)$$

    where M is the total number of candidate answers and C_i denotes their average log probabilities.

B.2 Features on Agent-Specific Criteria

- **Per-criteria scores by LLM judges** These features are introduced to measure the successfulness from human perspective (details in Table 5). We prompt an LLM to score the execution results based on these predefined criteria per subtask. For example, the execution result of an "add" excutor, "9 apples", might be evaluated with the following binary scores: {"accuracy of numerical value": 1.0, "sufficiency of context information": 0.0, "adherence to format": 1.0}. By assessing the correctness of the LLM executor's outputs and using these evaluations as features in training a verification agent, we ensure that the agent's detection is grounded in reliable, human-aligned guidelines.[6]

[6] Given that each subtask has a varying number of criteria, we create a one-hot vector to indicate the specific subtask to which each sample refers. This vector is then concatenated with a matrix containing the union of criteria columns across all subtasks, populated with the corresponding binary values from the execution result.

B.3 Other Features

- **Subtask type** is represented as one-hot encoding for all subtasks in our taxonomy.
- **Features on Plan Structures**
  - **Number of preceding subtasks** is the number of previous subtasks that this subtask is depending on (i.e., in-degree). We incorporate this feature to reflect the dependency information between subtasks within the plan.
  - **Source distance** measures the shortest chain length to reach the current subtask as a proxy for node importance.

C List of Prompts

We provide the prompts used for planning, agent execution, and criteria evaluation by external evaluators.

Prompt used for planning

```
You are a planner responsible for creating high-level plans to solve
any tasks using a set of agents.
Your goal is to break down a given task into a sequence of subtasks
that, when executed correctly by the appropriate agents, will lead
to the correct solution.

For each step in the plan:
1. Describe the subtask the agent must perform.
2. Provide a brief, self-contained description of the expected
inputs and outputs. Do not include any specific values or examples.
3. Provide a user prompt for each task that includes the expected
input and output information.

Represent your plan as a graph where each node corresponds to a
step, and each edge represents a dependency between two steps.
If a node requires the output from a previous node as an input,
ensure it is included in the edge list.
The output should be structured in the following JSON format:
{
“nodes”: <list of JSON nodes { “id”: <node id as integer>, “name”:
<assigned agent name>, “task”: <task instruction>,
“input”: <list of inputs>, “output”: <list of outputs>}>,
“edges”: <list of tuples [node_id, node_id]>
“user_prompts”: <list of strings per node>
}
Available agents: {agent taxonomy}

Examples
{plan demonstration examples}
-----------------------------------------
{task query}
```

Prompt used for agent execution and verbalized confidence

```
Use the following contextual information to answer: {context info}.
If contextual information is “None”, answer it without external
information.
JUST PERFORM WHAT YOU ARE ASKED TO DO, DO NOT ANSWER THE QUESTION,
JUST BECAUSE THE QUESTION EXISTS IN THE PROMPT.
Your answer should always be in JSON object format. {answer: <answer>,
confidence: <confidence>}.
-----------------------------------------
{subtask instruction prompt from the plan} + Also, provide how
confident you are in your answer.
If not, use your own memory to execute the prompt as best as you can.
If you do not know the answer, your confidence should be 0.0.
The answer format should be like {answer: <text>, confidence: <float
value between [0-1]>}.
```

Prompt used for using LLM evaluator with human-defined criteria

```
You’re a helpful assistant that evaluates an agent {agent}’s answer
in different criteria.
Your answer should always be in JSON format.
{‘criteria’:‘criteria score’}."
-----------------------------------------
Please evaluate the following agent’s answer to a user prompt with
the following context information.
If the context information is ‘None’, ignore and use your own
knowledge to answer.
Here are some examples to help you score the agent’s answer: {agent
examples}
The user prompt: {user}.
The context information: {context info}.
The agent: {agent}.
The agent’s input format: {agent input}.
The agent’s output format: {agent output}.
The agent’s answer: {answer}.
You should find the agent’s essential criteria to evaluate the answer
from {agent criteria list}
Then, score each criterion in a float value between [0-1] in the
’criteria score’ placeholder.
Your answer should look like:
{’criteria score’: {’<criteria 1>’: <float value between [0-1]>,
’<criteria 2>’: <float value between [0-1]>, ’<criteria 3>’: <float
value between [0-1]>, ’<criteria 4>’: <float value between [0-1]>,
’<criteria 5>’: <float value between [0-1]>} }
Your answer should always be in JSON object format:
{’criteria’:’criteria score’}.
```

D Agent Registry Curation using Chain-of-Thought Prompting

We provide the AI practioners to examine 50 samples resulting from one-shot promoting, and determine which agents are adequate for math reasoning application. As an example, we provide result for the task in Figure 2:

CoT planning result example

```
{
    "question": "Janet's ducks lay 16 eggs per day.
    She eats three for breakfast every morning and
    bakes muffins for her friends every day with four.
    She sells the remainder at the farmers' market daily
    for \$2 per fresh duck egg. How much in dollars
    does she make every day at the farmers' market?",
    "answer": "Janet sells 16 - 3 - 4 = <<16-3-4=9>>9
    duck eggs a day.\nShe makes 9 * 2 = $<<9*2=18>>18
    every day at the farmer's market.\n#### 18",
    "result": {
        "role": "assistant",
        "plans": {
            "step1": "Determine the total number of eggs
            Janet uses for breakfast and baking muffins.",
            "step2": "Subtract the number of eggs used
            for breakfast and baking muffins from the
            total number of eggs laid per day to find
            the number of eggs she sells.",
            "step3": "Calculate the total revenue by
            multiplying the number of eggs she sells
            by the price per egg."
        },
        "answers": {
            "step1": "Janet uses 3 eggs for breakfast
            and 4 eggs for baking muffins, so she uses
            a total of 3 + 4 = 7 eggs per day.",
            "step2": "Janet's ducks lay 16 eggs per day.
            After using 7 eggs, she has 16 - 7 = 9 eggs
            left to sell.",
            "step3": "Janet sells 9 eggs at $2 per egg,
            so she makes 9 * $2 = $18 per day at
            the farmers' market."
```

```
        }
    }
}
```

D.1 Candidate Agents for Other Tasks

We additionally present potential agent registries for the open-domain question answering task and fact-checking task (Table 4). We plan to expand VeriLA to build human-verifiable compound AI systems in these domains.

E Human-Defined Criteria for Agents

Table 5 lists evaluation criteria for each agent in our agent registry.

F Crowdsourced Annotation Procedure

We used MTurk platform to recruit crowdworkers for labeling the GSM8K dataset. For each instance, three labels were collected from workers who passed the qualification test, with an average pay rate of $14 per hour. For a subtask assigned to an agent, workers were asked to annotate whether the agent's answer was successful or not, given its role, input, and evaluation criteria. A sample interface illustrating the annotation process is shown in Figure 7.

G Experiment Details

G.1 Dataset Statistics

We partition each dataset to evaluate our approach. First, we reserve a held-out test set of tasks (math questions) used exclusively to evaluate the performances of aggregation metrics (Table 2). The remaining instances, which are decomposed into subtasks, are further split into train and test sets for training agent verifiers (Table 3). This hierarchical split ensures that agent verifiers are validated independently of the aggregator.

Table 2: Numbers of tasks in train-test split for aggregation.

Dataset	Train	Test
GSM8K	444	-
Date Understanding	137	35
Object Counting	158	40
Arithmetic Multistep	123	31

Table 3: Numbers of subtasks in train-test split for agent verifiers.

Dataset	Train	Test
GSM8K	778	195
Date Understanding	327	82
Object Counting	211	53
Arithmetic Multistep	578	145

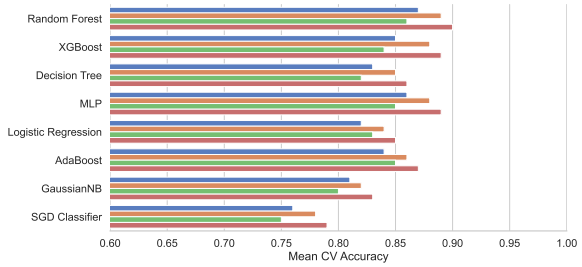

Figure 6: Verifier accuracy per model across datasets: average over 5-fold cross-validation.

G.2 Agent Verifier Accuracy Using Different Models

We train agent verifiers separately for each dataset using the following machine learning models: Logistic Regression [8], SGD Classifier [3], Decision Tree [4], Random Forest [26], AdaBoost [29], XGBoost [9], Gaussian Naive Bayes (GaussianNB) [18], and Multi-Layer Perceptron (MLP) [28]. Among all tested models, the random forest classifier with 100 tree estimators achieved the highest average accuracy among other models in four datasets (0.88). Figure 6 represents the accuracy of the verifier models, with results averaged over 5-fold cross-validation. Among the tested models, the random forest classifier with 100 tree estimators achieved the highest overall accuracy in four datasets.

Table 4: Candidate agents for open-domain question answering or fact-checking tasks.

Agent	Role	Input	Output	Output Format
Identify Query	Identify query for Wikipedia retrieval	Text	Text query	Text
Retrieve From Wikipedia	Retrieve from Wikipedia database	Text query	Most relevant paragraph	Text
Retrieve From DB	Retrieve paragraph from selected DB	Text query	Most relevant paragraph	Text
Brainstorm	Brainstorm ideas given input format	Text query	Brainstormed ideas	List
Rationalize	Generate explanation for given input	Text query	Explanation	Text
Classify	Assign category label to the input	Text query	Category-labeled JSON	JSON
Partition	Partition the problem into sub-problems	Text query	Partitioned sub-tasks	List
Merge	Combine multiple inputs into one	List of text	Combined text	Text
Compare	Compare two or more items	List of text queries	Comparison result	Text
Suggest	Generate improved ideas from input and instruction	Text query	Improved ideas	List
Transform	Transform text using suggested idea	Text query	Transformed text	Text
Extract	Extract smaller unit of text from input	Text	Extracted content	Text
Rate	Give rating to input	List of text	Sorted list by rating	List
Rank	Rank text based on criteria	List, criteria	Ranked list	List
Computation	Solve numerical computation task	Text query	Computed result	Text
Format Text	Generate formatted text from input and instruction	Text, instruction	Formatted text	Text
Prune	Remove redundant information	Text query	Pruned text	Text
Add	Add missing details to input	Text query	Completed text	Text
Correct	Correct errors in input text	Text query	Corrected text	Text

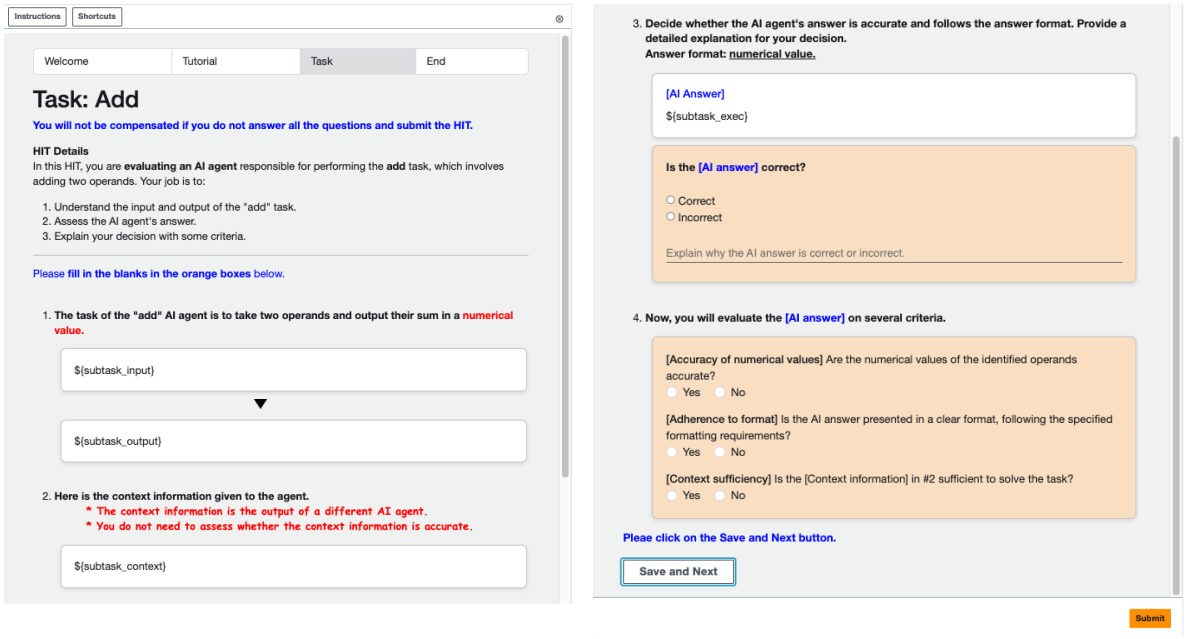

Figure 7: Example annotation for evaluation of LLM execution result of "add" subtask. We prohibit the users from moving on to the next page if they did not get the answer correct for questions in the tutorial.

Table 5: Human-designed agent criteria. Each agent's criteria are assigned by users based on their own experience performing the task using the agent registry. Thus, these criteria are grounded in human needs and are integrated into LLM evaluators, with their outputs used as part of our agent verifier features.

Subtask and Description	Essential Criteria
Identify Operands — Identify operands with text description of each operand	**Accuracy**: Are numerical values accurate? **Relevance**: Are all operands relevant? **Coverage**: Are all necessary operands identified? **Clarity**: Are operand descriptions clear? **Format Adherence**: Is the output correctly formatted?
Add — Add numbers or dates	**Accuracy**: Is the sum correct? **Format Adherence**: Is the output correctly formatted? **Context Sufficiency**: Is the context enough to solve the task?
Subtract — Subtract numbers or dates	**Accuracy**: Is the result correct? **Format Adherence**: Is the output correctly formatted? **Context Sufficiency**: Is the context enough to solve the task?
Multiply — Multiply numbers	**Accuracy**: Is the result correct? **Format Adherence**: Is the output correctly formatted? **Context Sufficiency**: Is the context enough to solve the task?
Divide — Divide numbers	**Accuracy**: Is the result correct? **Format Adherence**: Is the output correctly formatted? **Context Sufficiency**: Is the context enough to solve the task?
Filter — Filter a list based on a condition	**Relevance**: Are irrelevant items excluded? **Completeness**: Are all valid items included? **Format Adherence**: Is the output correctly formatted? **Context Sufficiency**: Is the context enough to solve the task?
Sort — Sort a list by an attribute	**Correctness**: Is the order accurate? **Completeness**: Are all items included? **Format Adherence**: Is the output correctly formatted? **Context Sufficiency**: Is the context enough to solve the task?
Convert Format — Convert input from one format to another	**Accuracy**: Was the conversion correct? **Format Adherence**: Is the output correctly formatted? **Context Sufficiency**: Is the context enough to perform the conversion?
Date Lookup — Identify year, month, and day from a natural language description	**Accuracy**: Is the date correctly identified? **Format Adherence**: Is the output correctly formatted? **Context Sufficiency**: Is the context enough to extract the date?