# IntellAgent: A Multi-Agent Framework for Evaluating Conversational AI Systems

Elad Levi Plurai eladl@plurai.ai Ilan Kadar Plurai ilan@plurai.ai

#### **Abstract**

Large Language Models (LLMs) are transforming artificial intelligence, evolving into task-oriented systems capable of autonomous planning, execution, and refinement. One of the primary applications of LLMs is conversational AI systems, which must navigate multi-turn dialogues, integrate domain-specific APIs, and adhere to strict policy constraints. However, evaluating these agents remains a significant challenge, as traditional methods fail to capture the complexity and variability of real-world interactions. We introduce **IntellAgent**, a scalable, opensource multi-agent framework designed to evaluate conversational AI systems comprehensively. IntellAgent automates the creation of diverse, synthetic benchmarks by combining policy-driven graph modeling, realistic event generation, and interactive user-agent simulations. This innovative approach provides fine-grained diagnostics, addressing the limitations of static and manually curated benchmarks with coarse-grained metrics. IntellAgent represents a paradigm shift in evaluating conversational AI. By simulating realistic, multi-policy scenarios across varying levels of complexity, IntellAgent captures the nuanced interplay of agent capabilities and policy constraints. Unlike traditional methods, it employs a graph-based policy model to represent relationships, likelihoods, and complexities of policy interactions, enabling highly detailed diagnostics. IntellAgent also identifies critical performance gaps, offering actionable insights for targeted optimization. Its modular, open-source design supports seamless integration of new domains, policies, and APIs, fostering reproducibility and community collaboration. Our findings demonstrate that IntellAgent serves as an effective framework for advancing conversational AI by addressing challenges in bridging research and deployment. The framework is available at https://github.com/plurai-ai/intellagent.

#### 1 Introduction

Large Language Models (LLMs) are revolutionizing the field of artificial intelligence by transitioning from static language processors to dynamic, task-oriented agents that can autonomously plan, execute, and refine their actions. These agents promise transformative applications across a wide range of domains, including healthcare [20, 1], finance [30, 26, 7], customer support [21, 14] and education [31, 29]. This evolution positions LLM agents as foundational technologies for reshaping human-computer interaction, enabling intelligent systems to tackle complex, real-world challenges with unprecedented efficiency and adaptability.

Among these advancements, conversational AI agents present a particularly demanding frontier. Unlike single-turn systems, these agents must navigate multi-turn dialogues, integrate domain-specific tools and APIs, and adhere to stringent policy constraints. The interplay of these requirements introduces a new level of complexity, where the cost of errors—ranging from inconsistent responses

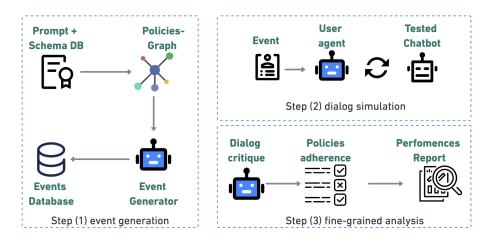


Figure 1: System diagram. (1) Given a chatbot prompt and a Schema DB, the system generates an event that targets a subset of policies, which includes a user request and a system DB state. (2) For each event the system simulates a conversation between the user and the chatbot. (3) A fine-grained report on the chatbot performances is generated.

to policy violations—can severely undermine reliability and trust. Reliability is not just desirable but critical, as it directly determines the feasibility of deploying such agents in real-world, high-stakes environments. Despite significant progress, evaluating conversational AI agents at this level of complexity remains a significant challenge [19, 9]. Traditional evaluation methods rely on static manually curated benchmarks [33, 15] that fail to scale or reflect the intricate dynamics of multi-turn interactions, policy adherence, and tool usage. These approaches often prioritize coarse-grained metrics, offering limited insights into agents' specific strengths and vulnerabilities. Consequently, existing methods leave critical gaps in both understanding and optimizing conversational agents for real-world applications.

In this work, we present **IntellAgent**, a scalable open-source multi-agent framework powered by AI agents, specifically designed to simulate and evaluate conversational AI agents comprehensively. IntellAgent represents a paradigm shift in evaluation by automating the generation of diverse, synthetic scenarios that rigorously test agents across multiple dimensions. Unlike existing approaches, IntellAgent leverages a novel pipeline that combines policy-driven graph modeling, realistic event generation, and interactive user-agent simulation to holistically assess agent performance, including a full spectrum of complexity levels, combinations of domain policies modeled through graph-based policy representations, and integration with API tools. The IntellAgent framework employs an automated process that involves constructing a policy graph to represent the relationships, complexity, and likelihood of various policies, generating events by sampling combinations of policies from the graph that align with real-world tasks and database states, and simulating interactions between a user agent and the chatbot. These simulations are then analyzed to provide fine-grained performance insights, identifying failure points, strengths, and opportunities for improvement. An overview of the system is illustrated in Figure 1, which outlines the key components of the IntellAgent pipeline.

Our study demonstrates the effectiveness and versatility of IntellAgent as a benchmarking tool for evaluating conversational AI agents. The results reveal a strong correlation between model performance on the IntellAgent benchmark and the  $\tau$ -bench [33], despite IntellAgent relying entirely on synthetic data. This validates IntellAgent as a robust alternative for evaluating conversational agents across diverse scenarios and challenges.

Key findings from our analysis indicate that model performance decreases with increasing complexity levels, but the rate of decline varies significantly across models. Additionally, our policy-specific evaluation uncovers significant variations in model capabilities across different policy categories. This highlights IntellAgent's ability to provide detailed diagnostic insights, empowering users to identify the most suitable configuration for their specific requirements.

This work makes several key contributions to the evaluation of conversational AI agents:

- A Scalable Multi-Agent Evaluation Framework: IntellAgent provides a robust and scalable multi-agent framework that automatically generates diverse, realistic scenarios tailored to address the unique challenges of conversational AI agents. By overcoming the limitations of small-scale, manually curated benchmarks, IntellAgent enables comprehensive and thorough evaluations.
- Fine-Grained Diagnostics with a Policies Graph: IntellAgent leverages a policies graph, inspired by GraphRAG [10], where nodes represent individual policies and their complexity, and edges denote the likelihood of co-occurrence between policies in conversations. This structure facilitates the generation of naturalistic user requests that span a wide range of policies and complexity levels, providing fine-grained diagnostic insights.
- An Open-Source and Extensible Framework: IntellAgent is released as an open-source framework, promoting reproducibility and community collaboration in advancing the evaluation and optimization of conversational AI agents. Its modular design supports seamless integration of new domains, including their respective policies, APIs, and database schemas. Additionally, the framework enables rigorous testing and optimization of custom conversational agents.

#### 2 Related Work

Recent advancements in using LLMs for synthetic data generation, automated evaluation have significantly influenced the development of AI systems. This section delves into these key areas, outlining existing methodologies and their limitations while highlighting how our approach advances the state of the art.

## 2.1 Synthetic Benchmarks

**Data generation.** Synthetic data generation using Large Language Models (LLMs) has become a transformative technique for advancing AI across various domains, including code generation [22, 32], mathematical reasoning [34, 18], text embedding [24], and text-to-image synthesis [5]. Synthetic data reduces the costs and time of human-annotated datasets while providing control over sample distribution, crucial for fine-tuning and optimizing performance in downstream tasks [16, 27]. Evaluating synthetic data focuses on two key metrics: *faithfulness* and *diversity*.

Faithfulness ensures synthetic data reflects real-world patterns and relationships, while diversity captures a wide range of scenarios to enhance model robustness and mitigate overfitting [17]. Achieving both metrics is challenging; recent research explores conditional prompting and multi-step generation to balance them. Conditional prompting improves diversity by defining attributes through condition-value pairs [12, 28]. Multi-step generation enhances coherence and domain coverage by decomposing tasks into smaller subtasks [8, 25, 13, 23], though these methods often require significant manual effort and may not scale well to complex domains.

Our approach automates synthetic dataset generation to ensure both faithfulness and diversity by using a policies graph inspired by GraphRAG [10], where nodes represent policies and edges capture their complexity relationships. This framework enables fine-grained generation across various combinations of policies, tools, and tasks, producing datasets that reflect application requirements while covering a diverse range of scenarios. It addresses challenges from simple tasks to complex edge cases, ensuring rigorous evaluation of agents under diverse conditions.

**Automated evaluation.** Synthetic datasets have become invaluable for evaluating retrieval-augmented generation (RAG) systems, offering metrics to measure retrieval quality, generation fidelity, and robustness. Frameworks like *RAGAS* [11] automate the evaluation of RAG pipelines, focusing on aspects such as retrieval accuracy and generation relevance. Unlike traditional metrics, RAGAS operates without reference answers, enabling broader applicability. However, it primarily targets isolated components and does not fully address the complexities of multi-turn dialogues or real-world conversational AI applications. This underscores the need for expanded evaluation frameworks that encompass diverse, dynamic use cases.

#### 2.2 Conversational AI Benchmarks

Evaluating conversational AI systems in real-world applications has been the focus of various benchmarks, each targeting specific capabilities. For instance,  $\tau$ -bench [33] assesses agents' ability to interact with users, adhere to domain-specific policies, and utilize API tools effectively, with simulations in domains like retail and airline customer service. However,  $\tau$ -bench is limited by its reliance on manual curation, with only 50 samples for airlines and 115 for retail, restricting scalability. Additionally, its evaluation focuses solely on coarse-grained end-to-end metrics, overlooking policy violations and dialogue flow errors, which limits comprehensive assessment.

The ALMITA benchmark [3] proposes a novel dataset and framework specifically tailored for evaluating tool-augmented conversational AI agents in customer support scenarios. It uses a combination of automated and manual processes to generate diverse and realistic conversations grounded in user-defined procedures. Despite its rigorous evaluation, ALMITA focuses primarily on customer support, and the generalizability to other domains remains an open question. Moreover, the reliance on manually curated samples, while ensuring quality, limits scalability.

The LTM Benchmark [6] effectively highlights limitations in conversational multitasking, especially with interleaved tasks. However, its reliance on predefined interaction structures limits its ability to capture the unpredictable and non-linear nature of real-world conversational flows, such as spontaneous topic shifts or revisiting earlier contexts. Similarly, E2E Benchmark [4] evaluates chatbot responses based on accuracy and usefulness, emphasizing conversational coherence in non-task-oriented interactions. Nevertheless, its lack of support for complex tool use and multi-turn interactions restricts its applicability to broader real-world contexts. The CURATe framework [2] addresses alignment challenges for personalized conversational agents by focusing on user-specific safety-critical contexts. While it introduces valuable techniques for multi-turn personalization, its emphasis on alignment rather than general performance testing narrows its scope.

Although these benchmarks provide valuable tools for assessing conversational AI systems, their reliance on manual curation limits scalability and adaptability to diverse real-world applications, making it challenging to generate the extensive datasets required for comprehensive evaluations. Our approach, in contrast, is fully automated, enabling the generation of diverse scenarios and dialogues at scale, thus supporting evaluations across varied domains. Additionally, our framework addresses a broader range of challenges, from simple tasks to complex edge cases, ensuring rigorous agent evaluation under diverse conditions. Unlike existing benchmarks that use coarse-grained metrics for overall performance, our method offers fine-grained insights by evaluating agents across all policy and tool combinations, identifying specific strengths and weaknesses.

# 3 Method

Our multi-agent system is illustrated in Figure 1. The system pipeline consists of the following steps: (1) The IntellAgent system receives a schema of the system database along with either a chatbot system prompt or a document outlining the company policies. Based on this input, the system constructs a policy graph (3.1.1). It then samples a list of policies from the graph at varying levels of complexity and generates an event addressing these policies (3.1.2). The event includes a scenario description with a user request and corresponding samples for the initial database state, ensuring the validity of the user requests. (2) The system simulates a dialog between the chatbot and a user agent using the information provided in the event (3.2). (3) Finally, a critique is provided with the dialog and provides an analysis of the chatbot's performances with respect to the event policies list (3.3).

#### 3.1 Event Generation

To address the challenge of developing advanced chatbots capable of interacting with a system database, the system must generate complex and natural user requests that cover various policies. Additionally, it should create an initial database state for the chatbot, ensuring that when the chatbot processes the user request and queries the database, it does not encounter failures.

An **IntellAgent event** is defined by the following components: (1) A list of policies. (2) A description of a user request that aligns with the specified policies. (3) The initial state of the chatbot's system database.

## **Algorithm 1** Event Policies Sampling

```
Require: A policy graph (G,E) with node weights \{n_g\}_{g\in G}, and a range n1 < n2 \in \mathbb{N} X = \emptyset \Rightarrow The events policies list while |X| < N do P = \emptyset e_w \sim \mathcal{U}(n_1,n_2), g \sim \mathcal{U}(G) \Rightarrow Sample the policy complexity and the initial policy while e_w > 0 do \Rightarrow Choose a neighbor h \in G with probability: P(X = h \mid Y = g) = \frac{E_{g,h}}{\sum_{h \in G} E_{h,g}} P \leftarrow P \cup \{h\} e_w \leftarrow e_w - n_h end while X \leftarrow X \cup \{P\} end while
```

#### 3.1.1 Policies graph

IntellAgent aims to generate a diverse set of events with varying levels of complexity. To create and complex user-chatbot interaction scenarios, IntellAgent constructs a policy graph. In this graph, nodes represent individual policies, and edge weights indicate the likelihood of two policies appearing together in the same interaction. Each node is also assigned a weight that reflects the complexity of its associated policy.

The graph is built through multiple queries to a large language model (LLM). First, the system extracts a list of policies from the prompt and assigns a difficulty ranking to each. Then, for every pair of policies, the LLM assigns a score (on a scale of 1–10) representing the likelihood of the two policies co-occurring in a conversation.

#### 3.1.2 Event generator

The complexity of an event is defined as the sum of the complexities of its policies. Given a policy graph with a set of policies G, weighted edges E and nodes complexity weights  $\{n_g\}_{g\in G}$ . the event generator aims to produce a **valid** set of policies that satisfies the following criteria:

- 1. The event complexity is uniformly distributed within a specified range.
- 2. The distribution of the first policy in the event is uniformly distributed across all possible policies.
- 3. For a given complexity level and initial policy, the distribution of the resulting policy list aligns with real-world distributions.

**Policies graph sampling.** To satisfy the outlined criteria, the IntellAgent sampling algorithm operates in batches. The process for each iteration is as follows: The complexity of events in the current batch is sampled first. The batch distribution is adjusted to ensure that the overall distribution of all generated events (including previous batches) remains uniform. Next, the initial policy for each event is sampled uniformly across all nodes in the policy graph. Then For each event, the system generates a policy path by performing a random walk on the graph. The walk terminates once the cumulative complexity of the visited nodes exceeds the sampled event complexity. An overview of the entire sampling method is provided in Algorithm 1.

This approach ensures that the generated events policies list maintains the desired complexity distribution and follows realistic transitions between policies as determined by the graph structure.

**Event generator agent.** The goal of the event generator agent is to create an event based on a given list of policies. The primary challenge is to generate a valid and consistent initial database state that the chatbot can interact with during the conversation. The agent's architecture is shown in Figure 4. To manage complex database schemas, the event generator agent first creates a symbolic

	<b>Uniform distribution</b>	Max Sampling	Weighted Sampling
Initial policy	The user cannot add travel insurance after the initial booking when <b>modifying</b> a reservation.	The user cannot add travel insurance after the initial booking when <b>modifying</b> a reservation.	The user cannot add travel insurance after the initial booking when <b>modifying</b> a reservation.
Policy 2	Cancelling a flight reservation: The refund will go to original payment methods in 5 to 7 business days.	The agent must first obtain the user id and the reservation id before <b>modifying</b> a flight reservation.	The agent must first obtain the user id and the reservation id before <b>modifying</b> a flight reservation.
Policy 3	Cabin class must be the same across all the flights in the same reservation	Basic economy flights cannot be modified. Other reservations can be <b>modified</b> without changing the origin, destination, and trip type.	The agent can only <b>cancel</b> the whole trip that is not flown.

Table 1: Comparison of random walk sampling strategies. (**Left**) Uniform sampling of the next node. (**Middle**) Selection of the next node based on maximal edge weight. (**Right**) IntellAgent weighted probability sampling, which balances diversity and alignment with real-world distributions.

representation of all the entities involved in the event. These entities are determined based on the provided chatbot prompt and the database schema. Typical entities may include users, products, reservations, etc.

The agent then iterates over these symbols, instantiating them by inserting the relevant rows into the database and replacing the symbolic variables with the corresponding data. This symbolic representation enables the agent to generate valid and consistent events, even across complex chatbot databases. Refer to Appendix B for an example of a generated symbolic representation and its corresponding database.

#### 3.2 Dialog simulation

For each event in the events database, IntellAgent simulates an interaction between a user and the chatbot being tested. Figure 5 provides an overview of the simulation architecture. The user agent is given the event details, which include the description of the event and all relevant information inserted into the chatbot's system database by the event generator agent. Additionally, the user agent is provided with the expected behavior of the chatbot at each step of the interaction, based on the event's policy list. The user has the option to terminate the interaction at any point, either when the chatbot successfully completes the task or if the chatbot fails to adhere to one of the policies and does not follow the expected behavior.

#### 3.3 Dialog Critique

The Dialog critique component is given the user-chatbot dialog and the chatbot system prompt to assess whether the reason for the dialog's termination, as provided by the user agent, is correct. If the reason is incorrect, the critique provides feedback to the user agent, and the dialog resumes. If the reason is correct, the critique then determines: (1) The subset of event policies that were tested during the dialog. (2) The subset of policies that the chatbot did not adhere to (this list may be empty). Using this information, a comprehensive report on the chatbot's performance is generated.

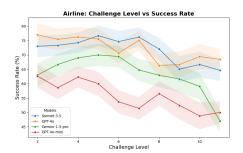




Figure 2: Model success rates across different challenge levels. While all models show reduced performance as the challenge level increases, they exhibit distinct patterns of decline, differing in both the onset level and the magnitude of the decrease.

Model	au-airline	IntellAgent-airline	au-retail	IntellAgent-retail
claude-3.5-sonnet	0.46	0.70	0.69	0.71
gpt-4o	0.44	0.70	0.51	0.68
gemini-1.5-pro	0.34	0.63	0.43	0.58
gpt-4o-mini	0.30	0.55	0.46	0.62
claude-3.5-haiku	0.28	0.53	0.44	0.56
gemini-1.5-flash	0.21	0.40	0.31	0.48

Table 2: Comparison of the success rates of various agent models utilizing function calling, evaluated on the  $\tau$ -bench and the IntellAgent benchmark. The results demonstrate a strong correlation between success rates across  $\tau$ -bench and IntellAgent benchmark.

# 4 Experiments

#### 4.1 Benchmark

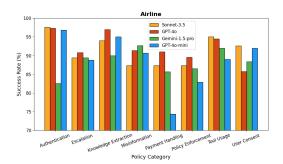
**Dataset construction.** To evaluate the performance of our system in real-world scenarios, we utilized the  $\tau$ -bench [33] environments. Specifically, we employed the benchmark's prompts, database schema, and tools for the two benchmark's environments: airline and retail. For each environment, the system generates a policy graph from the prompt and simulates **1,000 events**—a substantial increase that enables more fine-grained analysis compared to the original benchmark, which used 50 samples for airlines and 115 for retail. The complexity levels of the generated events range from 2 to 11. Table 1 presents a comparison of various random walk sampling strategies. Selecting the next node uniformly leads to unrelated policies, whereas choosing the next node based on maximal weight produces a cohesive cluster of policies. In contrast, IntellAgent-weighted probability sampling achieves a balance between diversity and alignment with the real-world distribution. Appendix B provides a detailed example of the event generation process.

**Tested agents.** We evaluated several state-of-the-art proprietary LLMs: GPT-4o, GPT-4o-mini, Gemini-1.5-pro, Gemini-1.5-flash, Claude-3.5-sonnet, and Claude-3.5-haiku. For all these models we employed the native tool-calling agent (supported by all the tested LLMs) with the tested environment system prompt. During each iteration, the model determines whether to send a message to the user or to call a tool.

**IntellAgent Implementation details.** The IntellAgent multi-agent system was implemented using the Langgraph<sup>1</sup> framework. GPT-40 served as the system's LLM model, used for the event generation, the user agent, and dialog critique. For full implementation details including all the system prompts, we provide the source code and documentation<sup>2</sup>

https://github.com/langchain-ai/langgraph

<sup>&</sup>lt;sup>2</sup>https://github.com/plurai-ai/intellagent



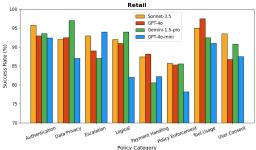


Figure 3: Comparison of the success rates of the top four models across various policy categories, highlighting that some categories are more challenging than others. Additionally, the relative performance order of different models varies across categories.

#### 4.2 Results

**Benchmarks comparison.** Table 2 presents a comparison of the tested model's success rates on the  $\tau$ -bench [33] and the IntellAgent benchmark. The Pearson correlation coefficients are **0.98** for the Airline environment and **0.92** for the Retail environment, highlighting a strong alignment in model performance across the two benchmarks. Notably, this strong correlation persists despite IntellAgent being generated exclusively with synthetic data.

**Models comparison.** Figure 2 illustrates the success rates of the top four models as a function of challenge level. As expected, model performance declines as the challenge level increases, though the pattern of decline varies across models. For instance, while Gemini-pro-1.5 significantly outperforms GPT-40-mini in the airline environment up to level 10, their performances converge at higher challenge levels. This highlights the value of IntellAgent's detailed analysis, enabling users to select the most suitable model based on the desired complexity the chatbot should handle.

**Policies comparison.** Figure 3 shows the performance of the top four models models across different policy categories. The relative ranking of models shifts across different categories. IntellAgent provides a detailed analysis of the specific policies where the tested chatbot may encounter difficulties.

It is also important to note that all models face challenges with *user consent policies*. This policy category is not assessed in the  $\tau$ -bench [33] since its evaluation focuses solely on the final state of the database.

#### 5 Conclusion

In this work, we introduced **IntellAgent**, a scalable, open-source multi-agent framework designed to comprehensively evaluate conversational AI systems. IntellAgent addresses the limitations of traditional evaluation methods by automating the generation of diverse, policy-driven scenarios and providing fine-grained diagnostics. By leveraging a graph-based policy model, realistic event generation, and user-agent simulations, IntellAgent captures the nuanced complexities of multi-turn dialogues, policy adherence, and tool integration.

Our findings demonstrate IntellAgent's ability to uncover critical performance gaps, offering actionable insights to optimize conversational agents for real-world applications. Its modular design supports extensibility, ensuring it remains relevant across diverse domains and use cases.

In future work, we plan to explore the benefits of incorporating additional real-world context into the environment, such as a small set of user-chatbot interactions. We hypothesize that this context could significantly enhance the policies graph quality by deriving edge weights and node challenge-level weights from the real-data distributions. Furthermore, we anticipate that this added context could improve the overall performance of the system database generation process.

#### References

- [1] M. Abbasian, I. Azimi, A. M. Rahmani, and R. Jain. Conversational health agents: A personalized llm-powered agent framework, 2023.
- [2] L. Alberts, B. Ellis, A. Lupu, and J. Foerster. Curate: Benchmarking personalised alignment of conversational ai assistants. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv preprint arXiv:2410.21159.
- [3] S. Arcadinho, D. Aparício, and M. S. C. Almeida. Automated test generation to evaluate tool-augmented llms as conversational ai agents. *arXiv preprint arXiv:2409.15934*, 2024.
- [4] D. Banerjee, P. Singh, A. Avadhanam, and S. Srivastava. Benchmarking llm powered chatbots: Methods and metrics, 2023.
- [5] J. Betker, G. Goh, L. Jing, TimBrooks, J. Wang, L. Li, LongOuyang, JuntangZhuang, JoyceLee, YufeiGuo, WesamManassra, PrafullaDhariwal, CaseyChu, YunxinJiao, and A. Ramesh. Improving image generation with better captions. 2023.
- [6] D. Castillo-Bolado, J. Davidson, F. Gray, and M. Rosa. Beyond prompts: Dynamic conversational benchmarking of large language models. arXiv preprint arXiv:2409.20222, 2024. 38th Conference on Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks.
- [7] H. Ding, Y. Li, J. Wang, and H. Chen. Large language model agent in financial trading: A survey, 2024.
- [8] N. Ding, Y. Chen, B. Xu, Y. Qin, S. Hu, Z. Liu, M. Sun, and B. Zhou. Enhancing chat language models by scaling high-quality instructional conversations. pages 3029–3051, Singapore, Dec. 2023.
- [9] H. Duan, J. Wei, C. Wang, H. Liu, Y. Fang, S. Zhang, D. Lin, and K. Chen. Botchat: Evaluating llms' capabilities of having multi-turn dialogues, 2023.
- [10] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson. From local to global: A graph rag approach to query-focused summarization. arXiv preprint arXiv:2404.16130, 2024. Microsoft Research.
- [11] S. Es, J. James, L. Espinosa-Anke, and S. Schockaert. Ragas: Automated evaluation of retrieval-augmented generation. *arXiv preprint arXiv:2309.15217*, 2023.
- [12] S. Gunasekar, Y. Zhang, J. Aneja, C. C. T. Mendes, A. D. Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi, A. Salim, S. Shah, H. S. Behl, X. Wang, S. Bubeck, R. Eldan, A. T. Kalai, Y. T. Lee, and Y. Li. Textbooks are all you need. *CoRR*, abs/2306.11644, 2023.
- [13] O. Honovich, T. Scialom, O. Levy, and T. Schick. Unnatural instructions: Tuning language models with (almost) no human labor. In *ACL*, pages 14409–14428. Association for Computational Linguistics, 2023.
- [14] K.-H. Huang, A. Prabhakar, S. Dhawan, Y. Mao, H. Wang, S. Savarese, C. Xiong, P. Laban, and C.-S. Wu. Crmarena: Understanding the capacity of llm agents to perform professional crm tasks in realistic environments, 2024.
- [15] W.-C. Kwan, X. Zeng, Y. Jiang, Y. Wang, L. Li, L. Shang, X. Jiang, Q. Liu, and K.-F. Wong. MT-eval: A multi-turn capabilities evaluation benchmark for large language models. In Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 20153–20177, Miami, Florida, USA, Nov. 2024. Association for Computational Linguistics.
- [16] E. Levi, E. Brosh, and M. Friedmann. Intent-based prompt calibration: Enhancing prompt optimization with synthetic boundary cases, 2024.
- [17] L. Long, R. Wang, R. Xiao, J. Zhao, X. Ding, G. Chen, and H. Wang. On llms-driven synthetic data generation, curation, and evaluation: A survey, 2024.
- [18] H. Luo, Q. Sun, C. Xu, P. Zhao, J. Lou, C. Tao, X. Geng, Q. Lin, S. Chen, and D. Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *CoRR*, abs/2308.09583, 2023.

- [19] A. V. Maharaj, K. Qian, U. Bhattacharya, S. Fang, H. Galatanu, M. Garg, R. Hanessian, N. Kapoor, K. Russell, S. Vaithyanathan, and Y. Li. Evaluation and continual improvement for an enterprise ai assistant, 2024.
- [20] N. Mehandru, B. Y. Miao, E. R. Almaraz, M. Sushil, A. J. Butte, and A. Alaa. Evaluating large language models as agents in the clinic. *npj Digital Medicine*, 7(1), Apr. 2024.
- [21] S. Rome, T. Chen, R. Tang, L. Zhou, and F. Ture. "ask me anything": How comcast uses Ilms to assist agents in real time. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR 2024, page 2827–2831. ACM, July 2024.
- [22] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. Canton-Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950, 2023.
- [23] F. Wan, X. Huang, T. Yang, X. Quan, W. Bi, and S. Shi. Explore-instruct: Enhancing domain-specific instruction coverage through active exploration. In *EMNLP*, pages 9435–9454. Association for Computational Linguistics, 2023.
- [24] L. Wang, N. Yang, X. Huang, L. Yang, R. Majumder, and F. Wei. Improving text embeddings with large language models. CoRR, abs/2401.00368, 2024.
- [25] R. Wang, W. Zhou, and M. Sachan. Let's synthesize step by step: Iterative dataset synthesis with large language models by extrapolating errors from small models. In *EMNLP (Findings)*, pages 11817–11831. Association for Computational Linguistics, 2023.
- [26] Y. Xiao, E. Sun, D. Luo, and W. Wang. Tradingagents: Multi-agents llm financial trading framework, 2024.
- [27] C. Xu, Q. Sun, K. Zheng, X. Geng, P. Zhao, J. Feng, C. Tao, and D. Jiang. Wizardlm: Empowering large language models to follow complex instructions, 2023.
- [28] R. Xu, H. Cui, Y. Yu, X. Kan, W. Shi, Y. Zhuang, W. Jin, J. C. Ho, and C. J. Yang. Knowledge-infused prompting: Assessing and advancing clinical text data generation with large language models. *CoRR*, abs/2311.00287, 2023.
- [29] S. Xu, X. Zhang, and L. Qin. Eduagent: Generative student agents in learning, 2024.
- [30] H. Yang, B. Zhang, N. Wang, C. Guo, X. Zhang, L. Lin, J. Wang, T. Zhou, M. Guan, R. Zhang, and C. D. Wang. Finrobot: An open-source ai agent platform for financial applications using large language models, 2024.
- [31] K. Yang, Y. Chu, T. Darwin, A. Han, H. Li, H. Wen, Y. Copur-Gencturk, J. Tang, and H. Liu. *Content Knowledge Identification with Multi-agent Large Language Models (LLMs)*, page 284–292. Springer Nature Switzerland, 2024.
- [32] Y. Yang, A. K. Singh, M. Elhoushi, A. Mahmoud, K. Tirumala, F. Gloeckle, B. Rozière, C. Wu, A. S. Morcos, and N. Ardalani. Decoding data quality via synthetic corruptions: Embedding-guided pruning of code data. *CoRR*, abs/2312.02418, 2023.
- [33] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.
- [34] Z. Yuan, H. Yuan, C. Li, G. Dong, C. Tan, and C. Zhou. Scaling relationship on learning mathematical reasoning with large language models. *CoRR*, abs/2308.01825, 2023.

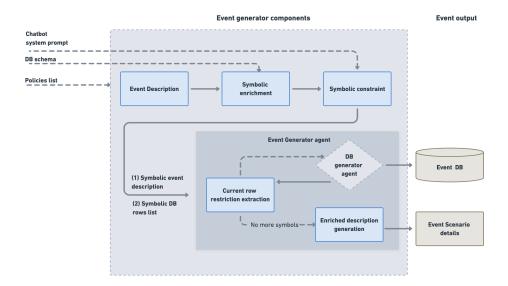


Figure 4: Event generator architecture overview.

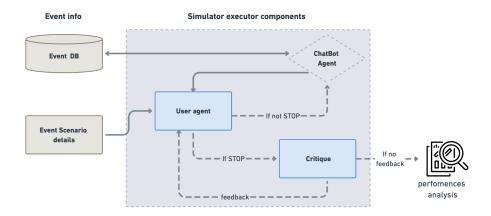


Figure 5: Simulator architecture overview.

# **Appendix**

# A Architecture details

This section offers further insight into the IntellAgent architecture. Figures 4 and 5 illustrate the architecture of the event generator and the simulator executor, respectively.

# **B** Complete Example of Event Generation

This section presents a comprehensive example of the generation process for *event 43* in the retail environment. Table 3 illustrates the entire process, starting from the requested challenge level, followed by extracting the event policies, generating the symbolic description, and finally obtaining

the detailed event scenario. Additionally, Figures 6, 7, and 8 showcase the generated event database for *event 43*.

Event Challange level	5
Event Policies	<ol> <li>Modify default user address: you have to authenticate the user identity by locating their user id via email, or via name + zip code</li> <li>You must deny any requests for tasks related to any other user.</li> </ol>
Symbolic description	User 'U1' initiates a conversation with the chatbot to update their default address from 'A1' to 'A2'. The chatbot authenticates 'U1' by verifying their email or name and zip code. During the conversation, 'U1' requests information about their spouse 'U2's recent order 'O1', claiming authority to access it. After this, 'U1' returns to their original request to update their default address.
Symbolic variables	'U1: The user initiating the conversation', 'U2: The spouse of the user', 'O1: A recent order made by the spouse', 'A1: The current default address of the user', 'A2: The new address the user wants to set as default'
Event Scenario details	User 'emma_jones_4829' initiates a conversation with the chatbot to update their default address from 'address1': '123 Elm Street', 'address2': 'Apt 4B', 'city': 'Seattle', 'country': 'USA', 'province': 'WA', 'zip': '98101' to 'A2'. The chatbot authenticates 'emma_jones_4829' by verifying their email or name and zip code. During the conversation, 'emma_jones_4829' requests information about their spouse 'liam_smith_9274's recent order '#09274620', claiming authority to access it. After this, 'emma_jones_4829' returns to their original request to update their default address.

Table 3: Event 43 generation steps

```
"emma_jones_4829": {
        "user_id": "emma_jones_4829",
        "name": "{'first_name': 'Emma', 'last_name': 'Jones'}",
        "address": "{'address1': '123 Elm Street', 'address2': 'Apt 4B', 'city': '
            Seattle', 'country': 'USA', 'province': 'WA', 'zip': '98101'}",
        "email": "emma.jones4829@example.com",
        "payment_methods": "{'paypal_4829103': {'source': 'paypal', 'id': '
            paypal_4829103'}, 'credit_card_1938472': {'source': 'credit_card', '
            brand': 'visa', 'last_four': '3847', 'id': 'credit_card_1938472'}}",
        "orders": "['#A1234567']"
   "user_id": "liam_smith_9274",
        "name": "{'first_name': 'Liam', 'last_name': 'Smith'}",
"address": "{'address1': '456 Oak Avenue', 'address2': 'Unit 12', 'city': 'Austin', 'country': 'USA', 'province': 'TX', 'zip': '73301'}",
        "email": "liam.smith9274@example.com",
        "payment_methods": "{'paypal_9274620': {'source': 'paypal', 'id': '
            paypal_9274620'}, 'credit_card_4620193': {'source': 'credit_card', '
            brand': 'visa', 'last_four': '4620', 'id': 'credit_card_4620193'}, '
            credit_card_567890': {'id': 'credit_card_567890', 'last_four': 1234, '
            brand': 'visa', 'source': 'card', 'balance': 50}}",
        "orders": "['#09274620']"
    }
}
```

Figure 6: Event 43 generated 'Users' dataset

```
{
   "1234567890": {
       "name": "Yoga Mat",
       "product_id": "1234567890",
       "variants": {
           "1011121319": {
               "item_id": "9876543210",
               "options": {
                  "color": "green",
                  "thickness": "5mm"
               "available": true,
               "price": 25.99
           },
           "1011121314": {
               "item_id": "9876543210",
               "options": {
                  "color": "blue",
                  "thickness": "5mm"
               "available": true,
               "price": 25.99
           }
       }
   },
"0987654321": {
       "name": "Dumbbell Set",
       "product_id": "0987654321",
       "variants": {
           "1011121319": {
               "item_id": "1234509876",
               "options": {
                  "weight": "20kg",
                  "material": "iron",
                  "color": "blue"
              },
               "available": true,
               "price": 89.99
          },
"1011121314": {
               "item_id": "1234509876",
               "options": {
                  "weight": "20kg",
                  "material": "iron",
                  "color": "green"
               "available": true,
               "price": 89.99
           }
       }
   }
}
```

Figure 7: Event 43 generated 'Products' dataset

Figure 8: Event 43 generated 'Orders' dataset