MermaidSeqBench: An Evaluation Benchmark for LLM-to-Mermaid Sequence Diagram Generation

Basel Shbita
IBM Research
San Jose, CA
basel@ibm.com

Farhan Ahmed
IBM Research
San Jose, CA
farhan.ahmed@ibm.com

Chad DeLuca
IBM Research
San Jose, CA
delucac@us.ibm.com

Abstract

Large language models (LLMs) have demonstrated excellent capabilities in generating structured diagrams from natural language descriptions. In particular, they have shown great promise in generating sequence diagrams for software engineering, typically represented in a text-based syntax such as Mermaid. However, systematic evaluations in this space remain underdeveloped as there is a lack of existing benchmarks to assess the LLM's correctness on this task. To address this shortcoming, we introduce MermaidSeqBench, a human-verified and LLM-synthetically-extended benchmark for assessing an LLM's capabilities in generating Mermaid sequence diagrams from textual prompts. The benchmark consists of a core set of 132 samples, starting from a small set of manually crafted and verified flows. These were expanded via a hybrid methodology combining human annotation, in-context LLM prompting, and rule-based variation generation. Our benchmark uses an LLM-as-a-judge model to assess Mermaid sequence diagram generation across fine-grained metrics, including syntax correctness, activation handling, error handling, and practical usability. We perform initial evaluations on numerous state-of-the-art LLMs and utilize multiple LLM judge models to demonstrate the effectiveness and flexibility of our benchmark. Our results reveal significant capability gaps across models and evaluation modes. Our proposed benchmark provides a foundation for advancing research in structured diagram generation and for developing more rigorous, fine-grained evaluation methodologies.

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in programming tasks such as code generation [1, 2], code documentation [3, 4], and coding assistants [5]. Furthermore, they show great promise in generating structured diagrams from natural language descriptions [6]. One type of structured diagram is a sequence diagram: a visual model used in software engineering that shows how objects, components, and processes interact with each other over time. Sequence diagrams are most commonly visualized in Unified Modeling Language (UML) [7], but can also be represented in a text-based syntax such as PlantUML [8] or Mermaid [9] (see Appendix A for more details). LLMs have proven to be proficient in generating such text-based based sequence diagrams [10, 11]. However, systematic evaluations for assessing an LLM's correctness in producing sequence diagrams remains largely underdeveloped. This has motivated research on developing evaluations for sequence diagram generation such as statistical method [12] or an LLM-as-a-judge [13].

Mermaid [9] is a diagramming tool that uses a Markdown-inspired syntax and is one such text-based method of representing sequence diagrams. Sequence diagrams textually represented in Mermaid are aptly named Mermaid sequence diagrams. Prior work has shown that LLMs are capable of

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling.

effectively producing Mermaid sequence diagrams [14]. However, evaluation in this space is even more underexplored as there is a severe lack of existing benchmarks or even public datasets to evaluate an LLM's Mermaid sequence diagram generation capabilities. This has driven our work to create a reproducible, scalable, and fine-grained evaluation framework for this task. Our key contributions are as follows:

- We introduce **MermaidSeqBench**, a human-verified and LLM-synthetically extended benchmark for assessing an LLM's capabilities in generating Mermaid sequence diagrams from textual prompts. This benchmarks consists of 132 samples and a systematic evaluation method using an LLM-as-a-judge to assess Mermaid sequence diagram generation across fine-grained metrics such as syntax correctness, activation handling, error handling, and practical usability.
- We perform initial evaluations on six models from three families and sizes (**Qwen 2.5** [15] at 0.5B and 7B, **Llama 3.1/3.2** [16] at 1B and 8B, and **Granite 3.3** [17] at 2B and 8B) on MermaidSeqBench using two large LLM judge models (**DeepSeek-V3** [18] at 671B and **GPT-OSS** [19] at 120B). This demonstrates both the effectiveness and flexibility of our proposed benchmark as our results reveal significant capability gaps across models and evaluation modes.

Our benchmark aims to provide a foundation for advancing research, not only in Mermaid sequence diagram generation, but also structured diagram generation as a whole. As a result, we open source MermaidSeqBench[1] and make the benchmark dataset publicly accessible[2] in hopes of encouraging further work in developing more rigorous, fine-grained evaluation methodologies in this space. Furthermore, although we focus on LLM-to-Mermaid sequence diagram evaluations, our methodology can be applied to other structural diagrams or even other textual representations such as PlantUML.

2 Related Work

Existing work on evaluating LLM generation of Mermaid sequence diagrams remains largely underexplored and very limited. Expanding to other forms of structural diagrams, Saxena et al. [14] and Guernsey [6] both explore methods on using LLMs to generate various structural diagrams in Mermaid syntax such as class, flow, and sequence diagrams. However, evaluations remain relatively simple through simple compliance or visualization checks which are not as robust or scalable.

In the broader space, most work on using LLMs to generate structural diagrams is typically represented textually in PlantUML [8]. There are several works on LLM generation of PlantUML class, flow, case, and sequence diagrams [6, 11, 20, 21]. Evaluations for these tasks provide more robust approaches. Rouabhia and Hadjadj [20] use statistical methods to validate the syntactic, structural, and behavioral consistency of the LLM generated UML class diagrams in similar way to existing code benchmarks such as HumanEval [22] or MBPP [23]. Ferrari et al. [12] utilize similar statistical methods to evaluate LLM generated UML sequence diagrams on categories such as completeness, correctness, and adherence to standards. Furthermore, Ahmed et al. [13] explore using an LLM-as-a-judge for evaluating the LLM-to-PlantUML sequence diagram task.

To the best our knowledge, no formal benchmark or even public dataset exists for evaluating an LLM's capabilities on producing Mermaid sequence diagrams. Our work is the first to introduce such a benchmark, public dataset, and systematic evaluation framework to evaluating this task in a similar manner to code generation benchmarks.

3 Methodology

Our methodology for constructing MermaidSeqBench proceeds in three stages: (1) initial dataset construction; (2) synthetic expansion using scalable LLM-based generation; and (3) systematic augmentation through rule-based variation. This pipeline enables us to create a benchmark that is both human-verified at its core and systematically extended for scalability and diversity.

[1] Code available at: `https://github.com/IBM/MermaidSeqBench-Eval`

[2] Dataset available at: `https://huggingface.co/datasets/ibm-research/MermaidSeqBench`

3.1 Initial Dataset Construction

We begin by constructing a small, high-quality seed set of ten Mermaid sequence diagrams. These were written manually by a subject matter expert (SME) given a set of natural language descriptions of different sequence flows. Each flow was manually verified to confirm syntax, semantic plausibility, and completeness. This curated core serves as the foundation of the benchmark.

3.2 Synthetic Expansion

To scale beyond the core set, we employ **Scalable Synthetic Data Generation (SDG)** [24], an open-source general-purpose LLM-wrapper framework designed for producing large quantities of synthetic data. In this stage, we use in-context examples of full flows of valid sequence diagrams in Mermaid syntax to guide the LLM in generating additional diagram flows. This allows us to systematically extend the benchmark while maintaining fidelity to the Mermaid syntax.

The SDG pipeline balances automation with control by leveraging prompts that encourage diverse structural patterns while adhering to syntactic rules. We used **Mistral Large** [25] (123B) as the primary generator. From the generated pool, 30 samples were selected and verified through two complementary mechanisms: (1) manual rendering via the Mermaid Live Editor [26] to ensure further syntactic correctness; and (2) manual verification with an SME to confirm completeness and adherence.

3.3 Rule-Based Variation Augmentation

Finally, we expand the resulting set through deterministic augmentation rules that produce approximately a four-fold increase in coverage. These rule-based transformations systematically alter both control flow structures and participant naming conventions, ensuring diversity in syntax without altering underlying semantics. In the first case, conditional constructs defined with `alt`, `else`, and `end` are programmatically detected and reordered, including support for nested alternatives. This reordering preserves the logical meaning of the branches while producing structurally distinct representations of the same flow. In the second case, participant identifiers are normalized into a canonical form, with substitutions consistently propagated across both the participant declarations and all subsequent message references. This guarantees syntactic validity while introducing surface-level variation in the representation of diagram entities.

3.4 Natural Language Descriptions

Each Mermaid diagram in the benchmark is paired with a structured natural language (NL) description capturing its *Purpose*, *Main Components*, and *Interactions*. This systematic format ensures that every diagram is grounded in a clear, unambiguous textual description, aligning with how interaction flows are typically documented in software design or requirements specifications. Together, these pairings yield a benchmark of 132 NL-Mermaid pairs in total. Examples of Mermaid syntax and rendered sequence diagrams can be found in Appendix A, while their corresponding natural language descriptions are detailed in Appendix B.

By combining human-verified samples, synthetically expanded instances, and rule-based variations, each paired with a structured natural language description, MermaidSeqBench is both faithful to real-world Mermaid sequence diagram usage and broad enough to expose LLM limitations across a wide range of structures. This integrated methodology ensures that the benchmark captures a wide range of diagramming scenarios.

4 Evaluation and Discussion

Our evaluation of MermaidSeqBench is aimed at demonstrating its utility for probing LLMs on structured sequence-diagram generation tasks. Our focus is to establish baseline results for state-of-the-art models while highlighting the flexibility of MermaidSeqBench when paired with independent LLM-as-a-Judge (LLMaJ) evaluators. This setup allows us to identify capability gaps that are often overlooked in standard text-generation benchmarks.

4.1 Experiments

We evaluate six models spanning three distinct families and parameter scales: **Qwen 2.5** [15] (0.5B and 7B), **Llama 3.1/3.2** [16] (1B and 8B), and **Granite 3.3** [17] (2B and 8B). This setup provides a balanced perspective on both lightweight and larger-scale variants across LLM families, enabling comparison of architectural differences as well as scaling effects.

For evaluation, we employ two large models as judges, namely **DeepSeek-V3** [18] (671B) and **GPT-OSS** [19] (120B). Each judge is prompted to assess generated Mermaid outputs. Given an input description (in natural language) and a corresponding reference diagram syntax, the judges rate candidate diagrams across six dimensions: *Syntax*, *Mermaid Only*, *Logic*, *Completeness*, *Activation Handling*, and *Error & Status Tracking*. Each dimension is scored on a scale from 0 to 1, allowing both fine-grained and aggregate comparisons of model performance.

4.2 Results

Table 1 presents the benchmark scores. Overall, the results reveal clear scaling effects across families: larger models within each family (7B/8B) consistently outperform their smaller counterparts (0.5B-2B). Among the strongest models, Qwen 2.5-7B and Llama 3.1-8B achieve the highest scores across most criteria, particularly on syntax, logical flow, and activation usage. Granite 3.3-8B follows closely but lags slightly on error handling and completeness.

Table 1: Model performance on MermaidSeqBench across six criteria, evaluated by two large LLM-as-a-Judge (LLMaJ) models. Higher is better.

	DeepSeek-V3 (671B)						GPT-OSS (120B)					
	Syntax	Mermaid Only	Logic	Completeness	Activation Handling	Error & Status Tracking	Syntax	Mermaid Only	Logic	Completeness	Activation Handling	Error & Status Tracking
Qwen 2.5-0.5B	58.90	77.12	36.93	44.39	26.52	38.07	48.95	65.45	13.91	18.41	13.85	15.90
Llama 3.2-1B	68.98	60.68	52.27	60.57	39.85	52.35	46.15	46.23	18.86	25.92	17.29	24.14
Granite 3.3-2B	75.27	90.98	70.23	74.55	63.71	71.86	39.60	78.59	34.11	46.50	29.01	53.97
Qwen 2.5-7B	91.29	95.98	87.23	88.90	79.55	81.70	85.97	97.05	70.56	77.37	64.04	69.81
Llama 3.1-8B	92.01	96.36	87.35	89.43	79.17	81.82	68.89	93.85	67.71	77.50	57.99	74.90
Granite 3.3-8B	86.97	94.13	83.03	83.75	74.13	76.97	65.15	88.35	58.90	65.08	47.08	63.24

Because all six evaluation criteria are evaluated by two independent large models serving as judges, the benchmark inherently captures cross-judge variability and robustness. DeepSeek-V3 assigns higher and more consistent scores across all criteria, while GPT-OSS tends to be stricter, especially for smaller models, leading to more pronounced gaps. This emphasizes the value of employing multiple LLMaJ evaluators, as they provide complementary perspectives and mitigate the bias of any single model. Overall, These findings confirm that MermaidSeqBench exposes meaningful differences across model families and sizes, and emphasizes the need to have a systematic, multi-faceted evaluation of LLMs on diagram generation tasks.

5 Conclusion

We introduced MermaidSeqBench, a benchmark for evaluating LLM capabilities in generating precise, structured, and logically consistent Mermaid sequence diagrams. Our method for creating this benchmark combines rule-based variation augmentation with natural language prompts, enabling systematic assessment across syntax, logic, completeness, activation handling, and error/status tracking. Initial experiments with six LLMs from three model families, judged by two large LLMs, reveal significant performance gaps across criteria and evaluation modes. These findings underscore the need for specialized benchmarks like MermaidSeqBench to expose fine-grained weaknesses and guide future model development.

References

[1] Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language models for code generation, 2024. URL https://arxiv.org/abs/2406.00515.

[2] Nam Huynh and Beiyu Lin. Large language models for code generation: A comprehensive survey of challenges, techniques, evaluation, and applications, 2025. URL https://arxiv.org/abs/2503.01245.

[3] Sayak Chakrabarty and Souradip Pal. Free and customizable code documentation with llms: A fine-tuning approach, 2024. URL https://arxiv.org/abs/2412.00726.

[4] Shubhang Shekhar Dvivedi, Vyshnav Vijay, Sai Leela Rahul Pujari, Shoumik Lodh, and Dhruv Kumar. A comparative analysis of large language models for code documentation generation, 2024. URL https://arxiv.org/abs/2312.10349.

[5] Xiaoyu Li et al. Conversational ai as a coding assistant. *arXiv preprint arXiv:2503.16508*, 2025. URL https://arxiv.org/abs/2503.16508.

[6] Grant Guernsey. Harnessing large language models for automated software diagram generation. Master's thesis, University of Cincinnati, 2025.

[7] Object Management Group. Unified modeling language (uml) specification, version 2.5.1. Technical Report formal/2017-12-05, Object Management Group (OMG), 2017. URL https://www.omg.org/spec/UML/2.5.1.

[8] Arnaud Roques. Plantuml: Generate diagrams from textual descriptions. https://plantuml.com/, 2025.

[9] Knut Sveidqvist, Sidharth Vinod, Ashish Jain, Neil Cuzon, Tyler Liu, Alois Klink, Reda Al Sulais, Nikolay Rozhkov, Justin Greywolf, Steph Huynh, Matthieu Morel, Marc Faber, Yash Singh, Nacho Orlandoni, Per Brolin, and Mindaugas Laganeckas. Mermaid: Javascript-based diagramming and charting tool. https://mermaid.js.org/, 2025.

[10] Munima Jahan, Zahra Shakeri Hossein Abad, and Behrouz Far. Generating sequence diagram from natural language requirements. In *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*, pages 39–48, 2021. doi: 10.1109/REW53955.2021.00012.

[11] Beian Wang, Chong Wang, Peng Liang, Bing Li, and Cheng Zeng. How llms aid in uml modeling: An exploratory study with novice analysts, 2024. URL https://arxiv.org/abs/2404.17739.

[12] Alessio Ferrari, Sallam Abualhaija, and Chetan Arora. Model generation with llms: From requirements to uml sequence diagrams, 2024. URL https://arxiv.org/abs/2404.06371.

[13] Khaled Ahmed, Jialing Song, Boqi Chen, Ou Wei, and Bingzhou Zheng. Mcet: Behavioral model correctness evaluation using large language models, 2025. URL https://arxiv.org/abs/2508.00630.

[14] Shobhit Sahai Saxena, Irshad Alam, Vaibhav Sharma, Umesh Vats, and Vijay Kumar Chundury. Dynamic creation of uml diagrams using generative AI. 2025.

[15] Qwen: An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

[16] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[17] IBM Granite Team. Granite 3.0 language models. *URL: https://github.com/ibm-granite/granite-3.0-language-models*, 2024.

[18] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, et al. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

[19] OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, et al. gpt-oss-120b gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

[20] Djaber Rouabhia and Ismail Hadjadj. Behavioral augmentation of uml class diagrams: An empirical study of large language models for method generation, 2025. URL https://arxiv.org/abs/2506.00788.

[21] Cecilia Eklund and Tom Jonsson. Benchmarking large language models in uml diagram generation from informal notations, 2025.

[22] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

[23] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

[24] foundation-model-stack. fms-dgt: Synthetic data generation for foundation models. https://github.com/foundation-model-stack/fms-dgt, 2025.

[25] Siddharth* Karamcheti, Laurel* Orr, Jason Bolton, Tianyi Zhang, Karan Goel, Avanika Narayan, Rishi Bommasani, Deepak Narayanan, Tatsunori Hashimoto, Dan Jurafsky, Christopher D. Manning, Christopher Potts, Christopher Ré, and Percy Liang. Mistral - a journey towards reproducible language model training, 2021. URL https://github.com/stanford-crfm/mistral.

[26] Knut Sveidqvist and Mermaid contributors. Mermaid live editor. https://mermaid.live, 2025.

A Sequence Diagrams

The UML sequence diagram in Figure 1 and the accompanying Listing 1 illustrate one of the test cases included in our benchmark dataset. The figure presents the rendered Mermaid sequence diagram describing the flow in uploading documents with secure storage, while the listing provides the exact Mermaid syntax used to generate it.

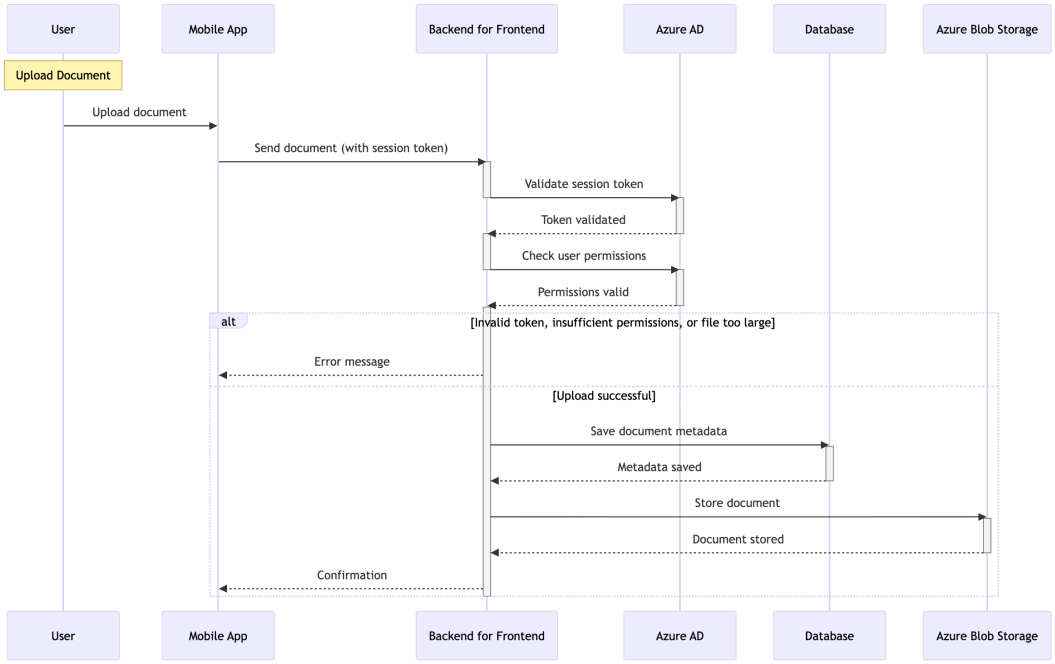

Figure 1: A UML sequence diagram from our benchmark, illustrating the "Uploading Documents with Secure Storage" flow. Participants include the User, Mobile App, Backend For Frontend (BFF), Azure AD, Database, and Azure Blob Storage. In this scenario, the User uploads a document through the Mobile App, which forwards the file and session token to the BFF. The BFF validates the token with Azure AD, checks the user's permissions, and, if authorized, records document metadata in the Database and securely stores the file in cloud storage (Azure Blob Storage). A confirmation is then returned to the app, while alternate paths handle errors for unauthorized access or oversized files.

```
sequenceDiagram
participant U as User
participant MA as Mobile App
participant BFF as Backend for Frontend
participant AAD as Azure AD
participant DB as Database
participant AS as Azure Blob Storage

Note over U: Upload Document
U->>MA: Upload document
MA->>BFF: Send document (with session token)
activate BFF
BFF->>AAD: Validate session token
deactivate BFF
activate AAD
AAD-->>BFF: Token validated
deactivate AAD
activate BFF
BFF->>AAD: Check user permissions
deactivate BFF
activate AAD
AAD-->>BFF: Permissions valid
deactivate AAD
activate BFF
alt Invalid token, insufficient permissions, or file too large
  BFF-->>MA: Error message
else Upload successful
  BFF->>DB: Save document metadata
```

```
  activate DB
  DB-->>BFF: Metadata saved
  deactivate DB
  BFF->>AS: Store document
  activate AS
  AS-->>BFF: Document stored
  deactivate AS
  BFF-->>MA: Confirmation
end
deactivate BFF
```

Listing 1: Mermaid syntax illustrating the "Uploading Documents with Secure Storage" flow, as rendered and detailed in Figure 1.

Our benchmark also includes more complex flows with multiple active actors. Figure 2 presents a sequence diagram demonstrating a user interaction with a Chatbot actor that may or may not invoke an additional actor (i.e., a Customer Support Agent). Listing 2 shows the Mermaid syntax corresponding to this diagram, which can be directly rendered to produce the figure.

```
sequenceDiagram
participant U as User
participant MA as Mobile App
participant BFF as Backend for Frontend
participant CB as Chatbot
participant CSA as Customer Support Agent

Note over U: User Query
U->>MA: Types a question or query
MA->>BFF: Sends query to BFF
activate BFF
BFF->>CB: Forwards query to Chatbot
activate CB
CB->>BFF: Sends initial response
deactivate CB
BFF->>MA: Forwards initial response
deactivate BFF
MA->>U: Displays initial response
Note over CB: Follow-up Questions
CB->>BFF: Asks follow-up questions
activate BFF
BFF->>MA: Forwards follow-up questions
deactivate BFF
MA->>U: Displays follow-up questions
U->>MA: Provides additional details
MA->>BFF: Sends additional details to BFF
activate BFF
BFF->>CB: Forwards additional details
deactivate BFF
alt Chatbot cannot resolve the issue
    CB->>CSA: Forwards escalated query
    Note over CSA: Agent Interaction
    CSA->>BFF: Interacts with user
    activate BFF
    BFF->>MA: Forwards agent interaction
    deactivate BFF
    MA->>U: Displays agent interaction
    U->>MA: Responds to agent
    MA->>BFF: Sends user response
    activate BFF
    BFF->>CSA: Forwards user response
    deactivate BFF
    CSA->>BFF: Provides solution or resolution
    activate BFF
    BFF->>MA: Forwards solution or resolution
    deactivate BFF
    MA->>U: Displays solution or resolution
end
Note over MA: Feedback
MA->>U: Prompts for feedback
U->>MA: Provides feedback
MA->>BFF: Sends feedback
activate BFF
BFF->>CSA: Forwards feedback
deactivate BFF
```

Listing 2: Mermaid syntax illustrating the "Chatbot Interaction for Customer Support" flow, as rendered and detailed in Figure 2.

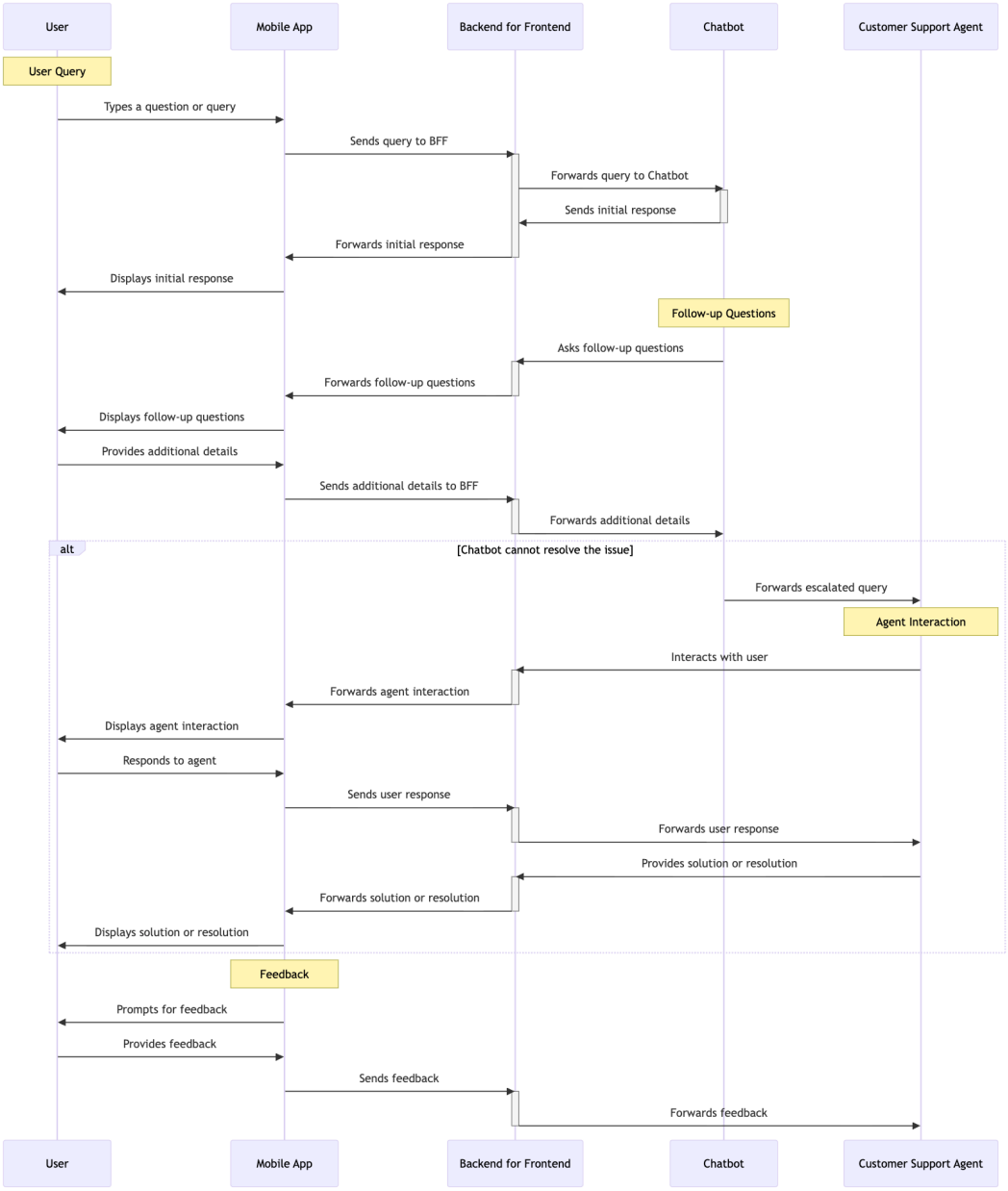

Figure 2: A UML sequence diagram from our benchmark, illustrating the "Chatbot Interaction for Customer Support" flow. Participants include the User, Mobile App, Backend For Frontend, Chatbot, and Customer Support Agent. In this scenario, the User submits a query through the Mobile App, which forwards it via the BFF to the Chatbot. The Chatbot provides initial responses and may request clarifications; if unable to resolve the issue, it escalates the conversation to a Customer Support Agent. The Agent then interacts with the User through the Mobile App to provide a resolution, after which the app collects feedback from the User.

B Natural Language Descriptions of Sequence Diagrams

For each Mermaid diagram provided in Appendix A, we include the corresponding natural language description that served as the input prompt for generation and evaluation. Each description systematically covers: (1) **Purpose:** the overall intent of the sequence diagram; (2) **Main Components:** the participants involved and their roles; and (3) **Interactions:** the ordered set of messages and control-flow constructs.

Listing 3 presents the natural language specification for the "Uploading Documents with Secure Storage" scenario (corresponding to the syntax provided in Listing 1 and rendered in Figure 1), while Listing 4 provides the specification for the "Chatbot Interaction for Customer Support" flow (corresponding to the syntax provided in Listing 2 and rendered in Figure 2).

```
Purpose: Uploading Documents with Secure Storage

Main Components: User, Mobile App, BFF, Azure AD, Database

Interactions:
1. User Action: User uploads a document (e.g., an ID or contract) through the mobile app.
2. Mobile App: Sends the document along with the session token to the BFF.
3. BFF Validation:
Validates the session token with Azure AD.
Checks user permissions to ensure they are authorized to upload documents.
4. Storage Process:
The BFF saves metadata about the document (e.g., file name, size, upload timestamp) in
the database.
The actual document is securely stored in cloud storage (e.g., Azure Blob Storage).
5. Response:
On successful upload, the BFF returns a confirmation to the app.
If the user is unauthorized or the file exceeds size limits, an appropriate error is
returned.
```

Listing 3: Natural language specification for the "Uploading Documents with Secure Storage" flow, corresponding to the syntax in Listing 1 and the rendered diagram in Figure 1.

```
Purpose: Chatbot interaction for customer support

Main Components: User, Mobile App, Chatbot, Customer Support Agent, BFF

Interactions:
1. User Query: The user types a question or query into the mobile app.
2. Chatbot Engagement: The mobile app sends the query to the Chatbot via the BFF.
3. Initial Response: The Chatbot processes the query and sends an initial response to the
 mobile app.
4. Follow-up Questions: The Chatbot may ask follow-up questions to better understand the
user's issue.
5. Escalation: If the Chatbot cannot resolve the issue, it escalates the query to a
Customer Support Agent.
6. Agent Interaction: The Customer Support Agent receives the escalated query and
interacts with the user through the mobile app.
7. Resolution: The Customer Support Agent provides a solution or resolution to the user's
 issue.
8. Feedback: The mobile app prompts the user to provide feedback on the support
experience.
```

Listing 4: Natural language specification for the "Chatbot Interaction for Customer Support" flow, corresponding to the syntax in Listing 2 and the rendered diagram in Figure 2.