NOVAID: Natural-language Observability Visualization Assistant for ITOps Dashboard Widget Generation

Pratik Mishra[1], Caner Gözübüyük[2], Seema Nagar[1], Prateeti Mohapatra[1], Raya Wittich[2], Arthur De Magalhaes[2]

[1]IBM Research
[2]IBM Software
{pratik-mishra, caner.goez}@ibm.com, {senagar3, pramoh01}@in.ibm.com, raya.wittich@de.ibm.com, arthurdm@ca.ibm.com

Abstract

Manual creation of IT monitoring dashboard widgets is slow, error-prone, and a barrier for both novice and expert users. We present NOVAID, an interactive chatbot that leverages Large Language Models (LLMs) to generate IT monitoring widgets directly from natural language queries. Unlike general natural language–to-visualization tools, NOVAID addresses IT operations–specific challenges: specialized widget types like SLO charts, dynamic API-driven data retrieval, and complex contextual filters. The system combines a domain-aware semantic parser, fuzzy entity matching, and schema completion to produce standardized widget JSON specifications. An interactive clarification loop ensures accuracy in underspecified queries. On a curated dataset of 271 realistic queries, NOVAID achieves promising accuracy (up to 94.10% in metric extraction) across multiple LLMs. A user study with IT engineers yielded a System Usability Scale score of 74.2 for NOVAID, indicating good usability. By bridging natural language intent with operational dashboards, NOVAID demonstrates clear potential and a path for deployment in enterprise ITOps monitoring platforms.

Introduction

The health and performance of modern IT systems are meticulously monitored by tracking various metrics and key performance indicators (KPIs) over time. This process, known as IT Operations (ITOps), involves a complex domain of data, which is often time-varying in nature and exposed through dynamic APIs. ITOps monitoring tools, such as Splunk (Splunk Inc. 2024), Datadog (Datadog Inc. 2024b), and Instana (Instana 2024), provide a means for site reliability engineers (SREs), developers, and other business professionals to track and assess system health through dashboards. While predefined IT dashboards are useful, they do not always cater to the specific needs of individual users. To address this, many existing tools (Instana 2024; Datadog Inc. 2024a; Grafana Labs 2025) offer the flexibility of creating custom dashboards that can be tailored to users' specific requirements by allowing them to add their desired widgets. These dashboards are crucial for better observability, offering users a centralized view of system health and performance while also enabling personalized visualization.

Extended paper at *github.com/DolceParadise/iaai-novaid*

However, the manual process of creating and configuring individual widgets for these custom dashboards is often a tedious and time-consuming task for both new and experienced users, and it can be prone to errors. The complexity of the widget creation flow also extends the onboarding time for new users (Elshehaly et al. 2020; Rossi, Tundo, and Mariani 2024). This challenge is compounded by ITOps domain-specific issues, such as specialized widget types (e.g., Service Level Objective charts), complex data access patterns through dynamic APIs, and the need to handle nuanced, context-rich queries with conditional filters.

We propose NOVAID (**N**atural-language **O**bservability **V**isualization **A**ssistant for **I**TOps **D**ashboards), an interactive chatbot powered by large language models (LLMs), to streamline the dashboard widget creation process. Our tool addresses the limitations of existing solutions by introducing a domain-specific, schema-aware approach to generate IT monitoring dashboard widgets from natural language queries. The tool parses user intent, extracts key elements, and interactively completes vague or incomplete queries through a multi-turn conversation, allowing users to quickly assemble custom dashboards without needing to understand the complex manual creation flow. The key contributions of our work are: (1) A novel AI application that tackles the significant and underexplored problem of natural language-to-dashboard widget generation in the ITOps domain. (2) A hybrid LLM-based architecture that combines an LLM-powered semantic parser with fuzzy entity matching to ensure precision and adherence to specialized IT monitoring data schemas. (3) An evaluation on a curated dataset of realistic ITOps queries and a user study with IT professionals, providing evidence of the system's effectiveness and its potential for real-world deployment.

Related Work

Natural Language to Visualization (NL2VIS) (Luo, Tang, and Li 2021; Srinivasan et al. 2021; Wang et al. 2022; Zhang et al. 2024) has attracted extensive attention, spawning both research prototypes and commercial products. Early systems (Gao et al. 2015; Setlur et al. 2016) pioneered mixed-initiative disambiguation and rapid statistical summarization. Toolkits like NL4DV (Narechania, Srinivasan, and Stasko 2020) provide end-to-end pipelines for parsing NL, inferring chart specifications, and rendering Vega-Lite

graphics. Commercial BI platforms have incorporated NL interfaces at scale (Jones 2014; Knight et al. 2022), allowing non-technical users to query enterprise datasets in plain English. Recent approaches (Chen et al. 2024; Maddigan and Susnjak 2023) leverage LLMs to interpret user queries and generate visualization code conditioned on a database schema. There is emerging work on end-to-end dashboard generation (Shi et al. 2020; Deng et al. 2022; Srinivasan and Stasko 2023; Dibia 2023) that can synthesize entire multi-chart dashboards from a single NL utterance.

However, existing NL2VIS solutions fall short when applied to the domain of ITOps and monitoring dashboards. These approaches—whether based on text-to-SQL translation or code generation for data retrieval and visualization—are not directly applicable to IT monitoring dashboards. IT monitoring platforms expose data through dynamic APIs. In this domain, visualization creation typically involves constructing a widget schema that defines the visual encoding, specifies parameters for API-based data retrieval, and determines the required time duration, reflecting the inherently time-series nature of IT data.

The following domain-specific challenges render existing NL2VIS techniques unsuitable for IT monitoring contexts:

1. Domain-Specific Widget Types: ITOps dashboards include specialized visuals like Service Level Objective (SLO) charts, which capture reliability metrics not commonly supported by general-purpose visualization libraries.

2. Complex Data Access Patterns: Data is accessed through APIs with dynamic parameter requirements. Even with fixed endpoints, constructing the correct query parameters—such as service names, application IDs, or metric types—demands precise interpretation of user intent.

3. Conditional and Contextual Filters: IT professionals pose nuanced, context-rich queries. For example:

> Show me the average latency over time for all HTTP calls made to the `catalogue` service, as well as the total number of calls for each call type in the `otel-shop` application.

Such queries require identifying entities (e.g., `catalogue` service, `otel-shop` application), relevant filters (e.g., HTTP calls), and aggregations across different scopes (e.g., average latency vs. total calls by call type).

NOVAID System Overview

NOVAID is designed to bridge the gap between natural language queries and fully functional, context-aware IT monitoring data visualizations. The system operates through a modular pipeline comprising tightly integrated components, each contributing to a seamless user experience—from understanding intent to rendering interactive dashboards. The process begins at a high level with the offline construction of a **knowledge base**, capturing global and instance-specific metadata. This foundation enables the subsequent inference of **key elements** from user queries.

To account for variations in terminology, wrong input, or incomplete input, we apply a robust **fuzzy matching** mechanism to align extracted elements with known entities in the system. Once these components are verified and completed, the system proceeds to **populate a structured JSON schema** which serves as the blueprint for rendering the desired widget.

Figure 1 shows the components and the interactions. Each of these stages is elaborated in the subsections that follow, demonstrating how the system transitions from user intent to a rendered, data-rich visual with minimal friction and maximum semantic alignment.

Knowledge Base Initialization

Our system relies on an online initialization process to construct a structured knowledge base that encapsulates essential domain and system-specific metadata. This step is critical for enabling consistent and intelligent behavior during system operation. The knowledge base comprises two distinct categories of information:

- **Type I: Global Knowledge.** This includes information that is agnostic to specific system instances and remains consistent across deployments. Examples include the set of supported widget types, metric names along with their allowed aggregation functions, and commonly used instance-invariant filter parameters. Since this data is static and universal, it is precomputed and stored as part of the system's embedded configuration.
- **Type II: Instance-Specific Knowledge.** This captures dynamic, deployment-dependent information that varies across different instances of the system. Examples include available service names, application identifiers, and endpoint names — entities that are tightly coupled with the runtime environment. As this data cannot be predefined, the system retrieves it at runtime via RESTful API queries to the target environment upon deployment.

Semantic Parser

Given a user query, our system extracts or infers (if the widget type is not mentioned in the query) the key elements required to construct a visualization widget. The most essential piece of information is the **widget type**, which determines how the data should be presented. A single widget can incorporate information from multiple data sources. For instance, a time series plot can display various series, each corresponding to a distinct data source.

A **data source** is defined as the combination of a *metric name*, an *aggregation function*, a set of *filter options*, and a *grouping criterion*. Filter options may range from simple attribute constraints to complex logical expressions.

For instance, consider the query: *"Show the mean latency of the catalogue service in the robot-shop application for HTTP calls, restricted to the top (bottom) five endpoints"*

This request translates into the following composite filter:

```
application.name = "robot-shop"
AND service.name = "catalogue" AND
call.type = "HTTP"
```

The grouping criterion specifies how the system should partition results and the number of outputs to return. In this example, the corresponding grouping criterion is:

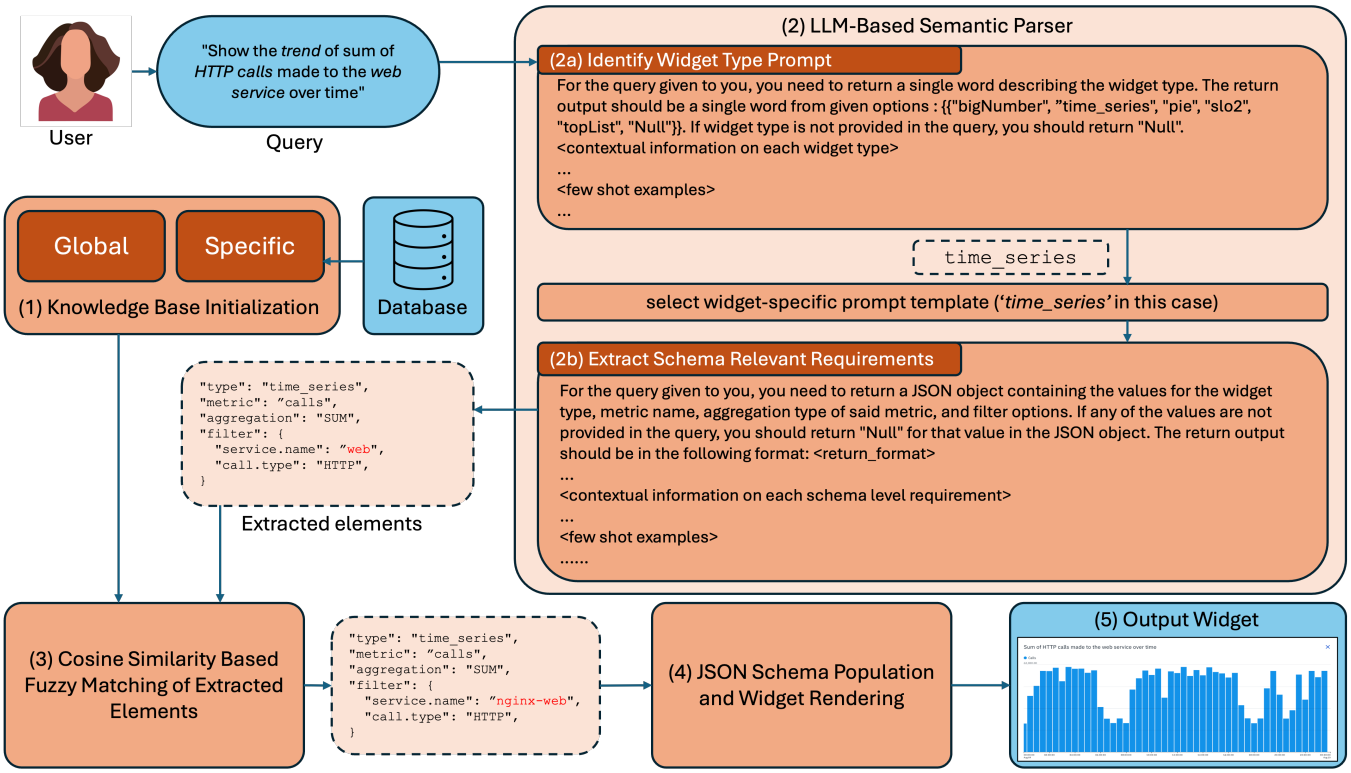

Figure 1: Framework of our proposed method of NOVAID for generating widgets for custom dashboards using LLMs

```
groupByTag = endpoint.name,
maxResults = 5, direction = DESC
```

We use a **two-pass process** to infer both the widget type and its corresponding data sources, motivated by the need to disambiguate user intent early and handle widget-specific schema differences.

- **First Pass – Widget Type Inference:** The LLM identifies the user's intent by selecting the appropriate widget type from a predefined knowledge base. This lightweight step provides essential context for the next phase.
- **Second Pass – Data Source Extraction:** With the widget type known, the LLM extracts the specific data sources required. This separation is necessary because data schema requirements vary—e.g., an *SLO widget* may need just a name, while a *time series widget* may require multiple metrics.

Fuzzy Matching of Extracted Elements

The extracted key elements from the user query may not always exactly match entries in the system's knowledge base. Users may have incomplete names, abbreviations, or slightly incorrect names. To handle this, we employ a **fuzzy matching strategy** based on *cosine similarity* between the extracted term and known values in the knowledge base. An **exact match** occurs if the cosine similarity score exceeds a defined threshold th, which is empirically decided. Based on the number and confidence of matches, we apply the following strategies:

1. Single High-Confidence Match (Auto-Correction) If there is only one candidate in the knowledge base whose similarity exceeds the threshold, we automatically replace the extracted value with this match.

- **Example:** The user refers to `qotd-service`, but the knowledge base contains only `qotd-web service`. As the similarity is high and unique, the system replaces `qotd-service` with `qotd-web service`.

2. Multiple Similar Matches (User Disambiguation) If multiple candidate values exceed the similarity threshold, we present them to the user as a list of suggestions. The user is then prompted to make a selection.

- **Example:** The user mentions `robot-shop service`, which does not exist in the knowledge base. However, two similar services, namely `robot-shop catalogue service` and `robot-shop shipping service`, are found. These are displayed to the user for manual selection.

3. Missing Key Element (Prompt for Completion) In cases where the user query omits an essential element, such as an aggregation method or specific filter value, we proactively detect the omission and prompt the user to choose from allowed values based on context.

- **Example:** In the query, *"I want to see the latency of the robot-shop application"*, no aggregation method is specified. The system detects this and prompts the user to se-

lect an appropriate aggregation (e.g., `mean`, `p95`, etc.) based on available options for the `latency` metric.

This fuzzy matching process increases system robustness to natural language variations and reduces the need for exact terminology, thereby improving usability for non-expert users.

JSON Schema Population and Widget Rendering

To enable dynamic dashboard generation, our system automatically constructs a JSON schema specific to each widget type by incorporating key parameters extracted from the preceding inference process, along with the user's selected time duration. This structured schema serves a dual purpose: it instructs the front-end on how to construct REST API requests for data retrieval, and it specifies the configuration required for rendering the corresponding widget. The JSON schema defines metadata, including the widget type and a configuration object. The configuration object specifies rendering options (e.g., line chart vs. bar chart), formatting preferences, and data source specifications. For SLO widgets, the data source is represented by the SLO name, whereas other widget types may reference multiple data sources.

Experiments and Results

Dataset

A major challenge in developing and evaluating NL interfaces for specialized domains is the lack of public datasets. To address this, we curated a custom dataset of 271 realistic, single-turn queries. These queries were gathered from real-world scenarios and discussions with IT professionals, simulating the types of requests an end-user would make and factoring in situations where a user makes a mistake or misses out some information. This meant that the dataset included queries where one or more key elements are intentionally absent in order to assess the "missing key element" functionality (discussed in System Overview section). For such cases, the corresponding fields in the ground truth dataset were set to *"null"*. The dataset is specifically structured to test the system's ability to extract and accurately map the following five key parameters to generate a valid widget JSON. **1. Widget Type:** The type of visualization requested (*e.g., bigNumber, time_series, topList*). **2. Metric:** The specific performance metric to be displayed (*e.g., calls, latency, erroneousCalls*). **3. Aggregation:** The function to be applied to the metric *(e.g., SUM, MEAN, PER_SECOND)*. **4. Tag Filter Expression:** Complex contextual filters based on key-value pairs (e.g., `'service.name': 'payment', 'call.type': 'HTTP'`). **5. Grouping:** The entity by which the data should be grouped, where applicable (e.g., `'groupbyTag': 'endpoint.name', 'direction': 'DESC', 'maxResults': 10`) Of the total 271 queries, 48 explicitly required grouping, presenting a more complex challenge for the model.

Evaluation Methodology

We evaluated the system's performance by measuring the accuracy of each extracted field (*widget type*, *metric*, *aggregation*, *tag filter expression*, and *grouping*) post knowledge-base cosine similarity comparison against the ground truth labels in our dataset. The primary metric for success is overall accuracy, defined as the percentage of queries where all five fields were extracted correctly, resulting in a fully valid and runnable widget JSON. This strict evaluation criterion ensures that a single error in any component invalidates the entire result, reflecting the practical requirements of generating production-ready dashboard configurations.

Model Details and Configuration

We tested our system with several open-source LLMs to demonstrate the robustness and independence of our pipeline from any single model. The LLMs evaluated were **Granite-3-3-8b-instruct** (Granite, IBM 2025), **Llama-3-3-70b-instruct** (Meta 2024), **Llama-4-Maverick-17b-128e-instruct** (Meta 2025), **Mistral-Small-3.1-24B-Instruct** (Mistral 2025), **gpt-oss-20b**, and **gpt-oss-120b** (OpenAI 2025). Each model was given a format-tailored system prompt and evaluated using a greedy decoding strategy with a maximum sequence length of 1024 tokens, avoiding the randomness introduced by sampling methods like top-k or nucleus.

Results

Table 1 summarizes the performance of various models on our evaluation bed. We initially tested zero-shot prompting to assess out-of-the-box model abilities, but the results were inconsistent. To improve robustness, we introduced few-shot prompting with 15 **diverse examples** in each prompt chosen to maximize coverage across different parameters. As shown in Table 2, few-shot prompting improved accuracy and stability, leading us to adopt it for all subsequent evaluations.

All models tested achieved high accuracy in identifying core components of a query, with Metric and Aggregation accuracies consistently above 92%, reaching a peak of 97.05% with the Llama-4-Maverick-17b-128e-instruct. The high accuracy in these fields confirms the viability of our domain-aware semantic parsing and LLM-based architecture for this complex task. The overall accuracy results, which are a highly conservative measure of performance, show that our system can successfully generate a complete and valid widget JSON for a significant portion of queries. Gpt-oss-120b achieved the highest overall accuracy at 67.0% (however, not a significant improvement when compared with granite-3-3-8b at p-value < 0.05), indicating its best ability to handle all five parameters simultaneously. A closer look at the data reveals that the primary challenge lies in extracting complex Tag Filter Expressions and, in particular, Grouping parameters. While Tag Filter Expression accuracy reached a respectable 80.07%, the Grouping accuracy varied, peaking at 64.58%. This is consistent with the nature of the task; grouping is often underspecified or implicit in a query, requiring more advanced reasoning to infer the user's intent. For example, a user asking for *"the services with the highest latency"* implies grouping by `service.name`, but the grouping field itself is not explicitly stated. This is where our proposed multi-turn clarification loop becomes essential for practical deployment, as it

Large Language Model	Accuracy (%)					
	Widget Type	Metric	Aggregation	Tag Filter	Grouping	Overall
Granite-3-3-8b†	86.72	92.25	92.62	75.65	**64.58**	54.24
Llama-3-3-70b†	**91.51**	92.62	92.99	76.01	**64.58**	58.67
Llama-4-Maverick-17b-128e†	89.30	**94.10**	**97.05**	**80.07**	56.25	64.21
Mistral-Small-3.1-24b†	89.30	**93.73**	**96.68**	**79.70**	56.25	63.47
gpt-oss-20b	88.93	92.99	95.20	76.01	60.42	60.52
gpt-oss-120b	**91.51**	92.62	92.99	76.01	**64.58**	**67.93**

Table 1: Performance of various Large Language Models on Widget Generation

† *Instruction-tuned model was used.*

Model	Zero-Shot	Few-Shot	Δ
Llama-3-3-70b†	62.96%	67.41%	**+4.45**
Granite-3-3-8b†	62.22%	73.88%	**+11.66**

Table 2: Tag Filter Expression accuracy improvement from Few-Shot prompting on a 242 query subset of the dataset

addresses the ambiguities that a single-turn parsing approach cannot resolve.

Deployment

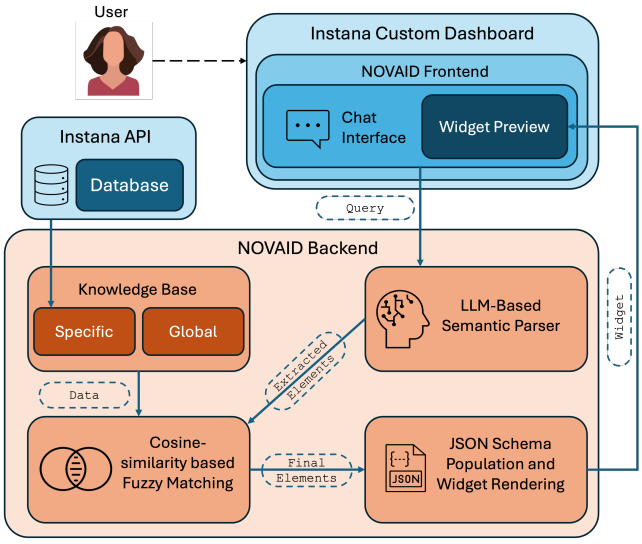

Figure 2: NOVAID Deployment and Integration with Instana

NOVAID is embedded as an emerging application within the **Instana platform** (Instana 2024), augmenting the traditional custom dashboard workflow with natural language interaction. Rather than requiring users to manually configure widgets through multi-step forms that are traditionally present and used in Instana Custom Dashboard's widget creation tool, NOVAID enables conversational widget authoring, lowering the barrier for SREs and developers to track

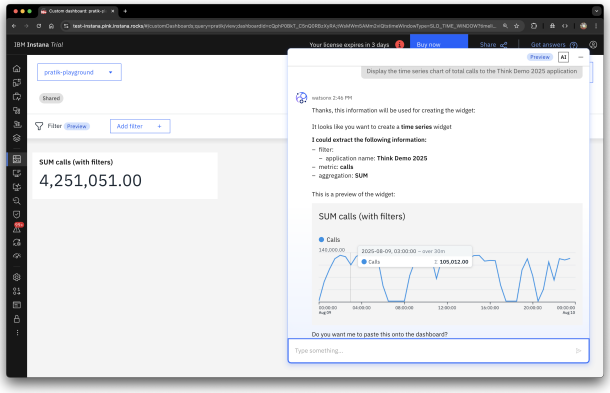

Figure 3: NOVAID Frontend with query and widget preview

and assess system health from monitoring data. Figure 2 goes over NOVAID's architecture and integration into the Instana ecosystem. The system consists of two main components: a **front-end interface** with chat and interactive widget preview (see Figure 3), and a **backend service** for query translation.

NOVAID Frontend

The front end is deployed on Instana's multi-region Kubernetes infrastructure, ensuring high availability and smooth integration with existing workflows.

The chat interface is the primary entry point for users, embedded directly within the Instana dashboard. Users express their requests in natural language—such as filters, aggregation strategies, or visualization preferences—and receive our tool's interpretations within the chat window. The interface supports multi-turn conversations, prompting for missing parameters and offering clarification options when user input is ambiguous or is missing key elements.

The widget preview renders an immediate interactive visualization inside the chat pane once NOVAID resolves a user's intent. Users can refine the preview through follow-up interactions, and once satisfied, confirm its addition to the main dashboard pane. This reduces trial-and-error in the manual widget builder and accelerates dashboard authoring

NOVAID Backend

The backend implements the natural-language-to-visualization pipeline, connecting the chat interface with Instana's monitoring services. Its responsibilities include (*i) intent parsing and parameter resolution*, *(ii) retrieving metadata and metrics from Instana APIs*, and *(iii) generating specifications for widget previews and confirmed dashboard widget*s. RESTful APIs expose this functionality for dynamic and automated integration. The backend runs in a per-tenant, containerized deployment on Instana's cloud infrastructure, enforcing isolation and authentication. Its modular microservices design allows incremental rollout of additional widget types (e.g. Histogram, Table) without disruption.

Operational Status and Rollout

NOVAID has been deployed in a test system for an internal user study for several months, demonstrating the potential of conversational workflows to reduce the dashboard customization efforts. Based on these evaluations, it is now being introduced as a public preview for Instana customers. IBM's LLM **Granite-3-3-8b-instruct** was chosen with its smaller size offering lower inference cost and latency compared to gpt-oss-120b, whose accuracy improvement is not statistically significant. Early users highlight the chat interface's value for intent clarification and the widget preview for rapid iteration, demonstrating both immediate utility and a clear trajectory to production readiness. Ongoing effort focuses on expanding supported widget types, improving disambiguation of domain-specific terminology, and validating scalability under production loads. Addressing these challenges forms the next step toward full-scale deployment.

User Study

To evaluate NOVAID's effectiveness and usability, we conducted a preliminary user study with IT support engineers from different domains (e.g., network, storage), including both experienced and novice users. Participants performed two tasks: manually creating widgets and using NOVAID for automatic creation, followed by a survey. They were encouraged to explore NOVAID beforehand and provided with task descriptions to guide their natural language queries. For each task, participants identified an IT fault and considered investigative information to aid remediation. Study data included widget correctness and survey responses. After completing the tasks, participants filled out a usability questionnaire and provided feedback. Table 3 presents widget correctness, rated on a 5-star scale (5 stars = 100%). Correctness improved with each interaction, reflecting a learning curve.

The second survey section used standard usability benchmark questions from (Lewis 2018) to assess NOVAID. Table 4 reports average scores from six participants. NOVAID achieved a System Usability Scale (SUS) score of 74.2, exceeding the benchmark of 68 (Sauro and Lewis 2016) and indicating good usability. Responses were on a five-point Likert scale (Albaum 1997), with odd items positive (higher = better) and even items negative (lower = better). Participants shared subjective feedback on their experience creating widgets. Overall sentiment was positive, with users finding NOVAID promising and easy to use.

First Widget	Second Widget	Third Widget
4	4	4
1	2	5
5	5	5
1	5	5
5	4	5
3	5	3
average correctness		
3.17	**4.17**	**4.5**

Table 3: Correctness of automatic widget creation

Statement	Average score
1. I think that I would like to use this system frequently.	**3.67**
2. I found the system unnecessarily complex.	**1.83**
3. I thought the system was easy to use.	**3.50**
4. I think that I would need the support of a technical person to be able to use this system.	**2.50**
5. I would imagine that most people would learn to use this system very quickly.	**4.50**
6. I thought there was too much inconsistency in this system.	**2.50**
7. I felt very confident using the system.	**3.83**
8. I found the system very cumbersome to use.	**2.00**
9. I found the various functions in this system were well integrated.	**4.33**
10. I needed to learn a lot of things before I could get going with this system.	**1.33**

Table 4: Usability Study

Conclusion and Future Work

In this paper, we introduced a method for automatically generating widgets for custom dashboards in IT monitoring tools via a conversational interface. Our approach performed well on curated queries, and a study confirmed users found the system intuitive and useful. The implementation and evaluation of NOVAID demonstrate the feasibility of generating dashboard widgets for ITOps. For future work, we will address limitations identified by user study participants First, we will improve key element extraction using dynamic few-shot learning. Second, we will refine prompts to reduce cost and model latency. Third, we aim to expand functionality by supporting more widget types and integrating additional data sources. Finally, we will enhance the user experience with error feedback and iterative widget refinement through natural language. Since NOVAID is ITOps tailored, it may not fully meet other domain-specific needs out-of-the-box, slightly limiting applicability.

References

Albaum, G. 1997. The Likert scale revisited. *Market Research Society. Journal.*, 39(2): 1–21.

Chen, N.; Zhang, Y.; Xu, J.; Ren, K.; and Yang, Y. 2024. Viseval: A benchmark for data visualization in the era of large language models. *IEEE Transactions on Visualization and Computer Graphics*.

Datadog Inc. 2024a. Create and Manage Custom Dashboards.

Datadog Inc. 2024b. Dashboards Overview - Datadog Documentation.

Deng, D.; Wu, A.; Qu, H.; and Wu, Y. 2022. DashBot: Insight-driven dashboard generation based on deep reinforcement learning. *IEEE Transactions on Visualization and Computer Graphics*, 29(1): 690–700.

Dibia, V. 2023. LIDA: A Tool for Automatic Generation of Grammar-Agnostic Visualizations and Infographics Using Large Language Models. In Bollegala, D.; Huang, R.; and Ritter, A., eds., *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, 113–126. Association for Computational Linguistics.

Elshehaly, M.; Randell, R.; Brehmer, M.; McVey, L.; Alvarado, N.; Gale, C. P.; and Ruddle, R. A. 2020. QualDash: Adaptable generation of visualisation dashboards for healthcare quality improvement. *IEEE Transactions on Visualization and Computer Graphics*, 27(2): 689–699.

Gao, T.; Dontcheva, M.; Adar, E.; and Karahalios, K. 2015. DataTone: Managing Ambiguity in Natural Language Interfaces for Data Visualization. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology (UIST '15)*, 117–126. New York, NY, USA: ACM.

Grafana Labs. 2025. Build your first dashboard. https://grafana.com/docs/grafana/latest/getting-started/build-first-dashboard/.

Granite, IBM. 2025. Granite-3.3-8B-Instruct Model Card. https://huggingface.co/ibm-granite/granite-3.3-8b-instruct.

Instana. 2024. Creating Custom Dashboards in Instana.

Instana. 2024. IBM Instana Observability.

Jones, B. 2014. *Communicating data with Tableau: designing, developing, and delivering data visualizations*. " O'Reilly Media, Inc.".

Knight, D.; Ostrowsky, E.; Pearson, M.; and Schacht, B. 2022. *Microsoft Power BI Quick Start Guide: The ultimate beginner's guide to data modeling, visualization, digital storytelling, and more*. Packt Publishing Ltd.

Lewis, J. 2018. The System Usability Scale: Past, Present, and Future. *International Journal of Human-Computer Interaction*, 1–14.

Luo, Y.; Tang, J.; and Li, G. 2021. nvBench: A large-scale synthesized dataset for cross-domain natural language to visualization task. *arXiv preprint arXiv:2112.12926*.

Maddigan, P.; and Susnjak, T. 2023. Chat2vis: Generating data visualizations via natural language using chatgpt, codex and gpt-3 large language models. *Ieee Access*, 11: 45181–45193.

Meta. 2024. LLaMA 3 (70B) Language Model. https://www.llama.com/models/llama-3/.

Meta. 2025. Llama-4 Maverick 17B-128E-Instruct Language Model. https://huggingface.co/meta-llama/Llama-4-Maverick-17B-128E-Instruct.

Mistral. 2025. Mistral-Small-3.1-24B-Instruct-2503 Language Model. https://huggingface.co/mistralai/Mistral-Small-3.1-24B-Instruct-2503.

Narechania, A.; Srinivasan, A.; and Stasko, J. 2020. NL4DV: A toolkit for generating analytic specifications for data visualization from natural language queries. *IEEE Transactions on Visualization and Computer Graphics*, 27(2): 369–379.

OpenAI. 2025. gpt-oss Language Model. https://openai.com/index/introducing-gpt-oss/.

Rossi, M. T.; Tundo, A.; and Mariani, L. 2024. Towards Model-Driven Dashboard Generation for Systems-of-Systems. https://arxiv.org/abs/2402.15257.

Sauro, J.; and Lewis, J. R. 2016. *Quantifying the User Experience: Practical Statistics for User Research*. Boston, MA: Morgan Kaufmann, 2nd edition. ISBN 978-0128023082.

Setlur, V.; Battersby, S. E.; Tory, M.; Gossweiler, R.; and Chang, A. X. 2016. Eviza: A Natural Language Interface for Visual Analysis. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*, 365–377. New York, NY, USA: ACM.

Shi, D.; Xu, X.; Sun, F.; Shi, Y.; and Cao, N. 2020. Calliope: Automatic visual data story generation from a spreadsheet. *IEEE Transactions on Visualization and Computer Graphics*, 27(2): 453–463.

Splunk Inc. 2024. Getting Started with Dashboards in Splunk.

Srinivasan, A.; Nyapathy, N.; Lee, B.; Drucker, S. M.; and Stasko, J. 2021. Collecting and characterizing natural language utterances for specifying data visualizations. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 1–10.

Srinivasan, A.; and Stasko, J. 2023. BOLT: A Natural Language Interface for Dashboard Authoring. In Hoellt, T.; Aigner, W.; and Wang, B., eds., *EuroVis 2023 – Short Papers*, 7–11. Eurographics.

Wang, Y.; Hou, Z.; Shen, L.; Wu, T.; Wang, J.; Huang, H.; Zhang, H.; and Zhang, D. 2022. Towards natural language-based visualization authoring. *IEEE Transactions on Visualization and Computer Graphics*, 29(1): 1222–1232.

Zhang, W.; Wang, Y.; Song, Y.; Wei, V. J.; Tian, Y.; Qi, Y.; Chan, J. H.; Wong, R. C.-W.; and Yang, H. 2024. Natural language interfaces for tabular data querying and visualization: A survey. *IEEE Transactions on Knowledge and Data Engineering*.