



Using Natural Language to Create Sophisticated Data Pipelines

Sierra Gong
Maximilian Knuth
Feliks Vaksman
David Turturean

Contents

1 Problem Statement and Background 1

1.1 About Nasdaq 1

1.2 Context and Problem Space 1

1.3 Project Goal and User Experience Design 1

2 Solution 2

2.1 System Design 2

2.2 Technical Considerations 3

2.3 User Interface: NASDAQ GenAI Terminal 4

2.4 Technology Advancement Beyond GPT-4.5 5

3 Impact and Business Value 5

4 Next Steps and Potential Improvements 6

5 Appendix 7

5.1 Langgraph Implementation 7

5.2 User Interface 8

5.3 Nasdaq’s Data Deployment Pipeline 8

5.4 Table Schema 9

1 Problem Statement and Background

1.1 About Nasdaq

Nasdaq is a leading global technology company that operates financial markets, curates market and alternative data products, and licenses advanced technology solutions supporting the capital markets ecosystem. Best known as the operator of the Nasdaq Stock Market, Nasdaq also provides a broad range of data and technology services to financial institutions, hedge funds, fintechs, and other market participants worldwide. Its business spans three main pillars: Markets & Listings (e.g., Nasdaq Stock Market, Nasdaq PHLX), Data & Index Products (e.g., Nasdaq Data Link, Nasdaq TotalView), and Technology Solutions (e.g., Nasdaq Matching and Marketplace technologies).

A critical part of Nasdaq's data services is Nasdaq Data Link, a platform that consolidates access to Nasdaq's market data and alternative data products. It offers customers unified access through APIs and web interfaces, making it a key touchpoint for data-driven clients such as analysts, quantitative researchers, and fintech developers.

1.2 Context and Problem Space

While Nasdaq has made major strides in democratizing data access through platforms like Nasdaq Data Link, the internal process of creating and maintaining data pipelines—required to transform raw data into curated, client-ready products—remains a significant bottleneck. Currently, this workflow is heavily reliant on specialized engineering teams. Business users, such as Data Product Managers, must submit data requests that often require multiple rounds of clarification and refinement with Data Engineers. Even for routine tasks like column renaming, field normalization, or time zone conversion, these requests can take weeks or even months to complete due to the rigid, manual handoffs and communication loops. Critically, revenue from these products is only realized after they are fully built and commercialized, meaning any delay in delivery directly postpones financial return. This inefficiency results in long lead times, underutilized engineering resources, and reduced responsiveness to client demands.

Estimates from this project suggest that only about 25% of incoming data product requests can be addressed with current engineering capacity. The remaining 75% are either delayed or dropped, leading to missed opportunities, unmet client needs, and deferred revenue. This shortfall not only frustrates customers—such as hedge funds, banks, and institutional investors—but also constrains Nasdaq's ability to scale its growing data product portfolio in response to rising market demand. Furthermore, the reliance on manual workflows introduces operational fragility, limits flexibility, and impedes experimentation, all of which hinder Nasdaq's strategic agility in an increasingly competitive landscape.

1.3 Project Goal and User Experience Design

This project presents a proof-of-concept for a Text-to-Data Pipeline solution—an agentic AI system named Nas1—designed to enable non-technical users to define and execute sophisticated data pipelines using natural language instructions. Rather than relying on manual coding or traditional ETL tools, Nas1 employs a modular multi-agent architecture powered by large language models and task-specific agents. The system interprets user instructions, validates inputs, routes tasks to specialized agents such as schema extractors, code generators, and execution managers, and returns fully processed datasets in near real-time.

The target users of this system are internal Nasdaq Data Product Managers who frequently coordinate with engineering teams to deliver custom data products to institutional clients such

as hedge funds, investment banks, and research analysts. These users typically lack programming expertise but have a deep understanding of business requirements and data context. As such, a user-relevant interface was critical to the project’s design. Instead of implementing this tool as a backend-only system or a rigid form-based interface, we chose to deliver the experience through a chat-based interaction flow. This decision reflects the need for flexible, iterative clarification, enabling users to express ambiguous or incomplete requests in plain English and refine them through conversation with the system.

By transforming the internal data product request workflow, this proof-of-concept aims to reduce engineering effort on routine data preparation tasks, increase the throughput of product delivery, and shorten time-to-market for valuable client-facing assets. The broader objective is to modernize Nasdaq’s internal operations and unlock substantial business value.

However, this challenge also presents a compelling opportunity for transformation. By introducing an agentic AI system that enables non-technical users to define and execute data pipelines through natural language, Nasdaq can significantly streamline its internal workflow. This system would automate routine data preparation tasks, validate schema and logic in real time, and deliver production-ready outputs without requiring continuous engineering involvement. In doing so, Nasdaq could dramatically increase engineering throughput, reduce turnaround time for custom data products, and enable faster monetization of data assets—ultimately unlocking millions of dollars in additional annual revenue while enhancing customer satisfaction and organizational scalability.

2 Solution

As part of this proof-of-concept, we used three synthetic datasets and one pdf file noting table schema (see Figure 4) as source files and implemented two core subagent functionalities: a date transformation agent and a join tables agent. These functionalities were selected based on areas frequently encountered by Nasdaq’s internal data product team and serve to validate the feasibility of the proposed system. Both subagents were developed with modularity in mind, allowing for future scaling and integration with Nasdaq’s internal databases and production workflows.

We named our solution *Nas1*, reflecting its role as Nasdaq’s first conversational data pipeline assistant. At a high level, *Nas1* enables non-technical users to describe data transformation tasks in plain English. The system then interprets these instructions, identifies the required operations, validates technical feasibility, generates executable Python code, and runs this code in a secure environment. This transforms what was previously a manual, engineering-intensive process into an automated workflow that is fast and accessible through natural language interaction.

2.1 System Design

The architecture of *Nas1* is implemented as a modular, multi-agent system using the LangGraph framework, which enables the orchestration of natural language data pipeline requests into executable code. The high-level agentic workflow is illustrated in Figure 1, showing how the system dynamically routes user instructions through a sequence of task-specific agents, validation stages, and code execution modules.

The workflow begins with the `first_run_router` and `task_classification` agents, both of which invoke the DeepSeek V3 language model to interpret the user’s natural language input and classify the intended data operation. DeepSeek is used here to ensure high-quality intent recognition, even when queries are ambiguous, multi-intent, or informally phrased. If the

query contains multiple tasks, the `split_query_agent` is called—again leveraging DeepSeek—to decompose the input into distinct, logically separated subtasks that can be handled in parallel or sequentially.

Once each task is identified, the `lm_first_pass` agent uses DeepSeek to extract structured parameters such as table names, column references, and transformation intents from the raw text. These extracted parameters are passed to the `task_router`, which dispatches the request to the appropriate specialized subagent.

For example, datetime conversion tasks are routed to the `router_timezone` agent. This agent performs parameter search and validation using schema lookups, as shown in Figure 1, powered by a Retrieval-Augmented Generation (RAG) layer. The RAG layer includes components such as `tz_rag_lookup` and `tz_extractor`, and its role varies depending on the stage at which it is invoked. In some cases, it suggests possible values for missing fields from original user query to support human clarification. In others, it retrieves and verifies that the specified tables or columns exist and are of the correct type, ensuring that the transformation logic aligns with schema constraints.

Similarly, join operations are handled by the `router_join_table` agent, which conducts metadata retrieval via `join_table_rag_lookup` and performs structural validation through `join_table_check`. These agents ensure that referenced tables and columns are compatible and that the join logic is semantically valid.

If ambiguities or missing fields are encountered at any stage, the system invokes the `ask_human` and `human_rout` agents to request clarification. During this step, the system passes the current context, validation errors, missing fields, and relevant RAG-based suggestions to DeepSeek via the `generate_smart_suggestion` function. This allows the model to generate context-aware prompts that inform the user which specific fields require clarification, enhancing user guidance and reducing interaction overhead.

If the `split_query_agent` detects that a user's input contains multiple distinct tasks, the system records the current workflow context and loops back to the `lm_first_pass` agent to extract the necessary parameters for each additional subtask—again using DeepSeek for consistent parsing quality. Each subtask is then processed through the same validation and clarification sequence as the initial task. If the user provides mid-process revisions, DeepSeek is called again to reinterpret the updated query and regenerate a revised task breakdown via the `split_query_agent`, ensuring synchronization across all downstream agents.

After all validations are complete, the `code_generation` agent uses DeepSeek one final time to generate executable Python code using predefined templates. This step relies on DeepSeek's ability to translate structured task definitions into syntactically correct, semantically aligned code. The code is executed in a secure environment, transforming the data according to the user's request. The workflow concludes at the `end_` node, returning the processed dataset or execution logs to the user.

This graph-based architecture supports flexible branching, iterative refinement, and human-in-the-loop interaction while maintaining full traceability across the pipeline. The design carefully balances automation with control, providing the efficiency of AI-driven execution along with the reliability of explicit validation at every step.

2.2 Technical Considerations

The technical architecture of Nas1 was guided by four core considerations: data privacy, model selection, system modularity, and scalability toward production integration. This section reflects on these considerations and evaluates how our design choices address the project's technical grading criteria.

Data Privacy and Security

Given Nasdaq's data sensitivity, we deliberately chose to develop and validate the system using synthetic datasets that mirrored the structure of real internal data. This allowed us to demonstrate the system's functionality without exposing any client or proprietary information. No production data was used in model prompts, retrievals, or executions. Additionally, the system was designed to execute generated code in a secure, sandboxed environment to prevent the risk of untrusted code execution. These safeguards ensured full alignment with data privacy and security best practices throughout development.

Model Selection and Cost-Benefit Tradeoff

Our choice of DeepSeek-V3 as the primary language model was driven by its balance between task performance, latency, and cost. While larger models like GPT-4 or Claude 3 could provide higher raw language capabilities, their operational costs and inference latency were impractical for a scalable enterprise application. DeepSeek-V3 provided sufficient language understanding and task decomposition capabilities at a fraction of the token cost, making it a pragmatic choice for balancing prototype fidelity with future production feasibility.

For the retrieval layer, we selected OpenAI's `text-embedding-3-small` due to its robust performance in generating dense vector embeddings and its integration with the LangChain and FAISS ecosystem. This enabled us to build a Retrieval-Augmented Generation (RAG) layer that supported real-time metadata retrieval, improving semantic grounding without requiring model retraining.

Prompt Engineering and Iterative Refinement

Prompts were designed to be modular and interpretable, following an iterative refinement process based on system testing. For example, the timezone extraction prompt was revised multiple times to improve parameter recognition and response consistency. Each iteration introduced structured output requirements and incorporated user clarification loops to handle ambiguous inputs. This iterative prompt design helped mitigate hallucinations and ensured that the language model responses aligned with the system's downstream execution logic. We provide a representative example of a refined system prompt in the appendix.

Modularity and Extensibility

Nas1 was architected as a modular, agent-based system to support future scaling through simple new field, node and edge definition. This design allows individual agents—such as task classifiers, metadata retrievers, or code generators—to be reused, extended or replaced independently. For example, the current RAG index uses synthetic metadata, but the architecture can be extended to connect with Nasdaq's production schema repositories. Similarly, external API-based LLMs could be replaced with self-hosted models to improve privacy and reduce dependency on third-party services.

In summary, our design choices reflect a balanced consideration of security, cost, scalability, and user reliability. Nas1 demonstrates how an agentic, modular system can operationalize natural language instructions while providing a clear path toward production-scale deployment.

2.3 User Interface: NASDAQ GenAI Terminal

To make the system accessible to non-technical users, we developed the NASDAQ GenAI Terminal—a web-based, chat-style interface that allows Nasdaq's Data Product Managers to define data tasks using plain English. These users are experts in data products but typically do not write code, making a conversational interface the ideal choice.

Instead of filling out rigid forms or relying on engineers, users interact with Nas1 through a terminal-like chat window that feels familiar to finance professionals. They receive immediate system feedback, including clarifications and confirmations, ensuring they stay in control

throughout the process.

The application runs on a Python-based backend and a React frontend, connected via real-time WebSocket communication. This setup provides instant responses, interactive confirmations, and live progress updates without page reloads.

As shown in Figure 2, the system walks users through each step—from understanding their request to validating details and running the data transformation. This human-in-the-loop interaction ensures that even complex operations feel easy and reliable, making data pipeline creation faster and more accessible for business users.

2.4 Technology Advancement Beyond GPT-4.5

While GPT-4.5 offers strong general-purpose capabilities, our solution goes beyond by introducing task-specialized agents, dynamic retrieval mechanisms, and system-level traceability.

Nas1 decomposes user queries into modular tasks handled by dedicated subagents, improving reliability and reducing hallucinations. Unlike a single large model, this architecture allows each agent to apply domain-specific logic, such as schema validation or code generation.

A key differentiator is the integration of a Retrieval-Augmented Generation (RAG) layer. This enables the system to dynamically retrieve up-to-date metadata and schema information, ensuring that pipeline generation is grounded in actual data structures rather than relying solely on the model's pre-trained knowledge.

To ensure transparency, Nas1 logs all routing decisions, validations, and generated outputs. This traceability supports auditability and builds user trust, which is critical in regulated environments like financial services.

3 Impact and Business Value

Nasdaq's data deployment pipeline spans multiple stages—from ingesting raw data to transforming, productizing, and deploying client-facing data products. As illustrated in Figure 3, business value is only realized once a product has been fully onboarded, processed, and made accessible to clients. However, several early-stage onboarding tasks—such as data parsing, cleaning, reformatting, and normalization—still rely heavily on manual engineering, creating bottlenecks that delay time-to-market and limit overall throughput.

Nas1 addresses these inefficiencies by automating critical onboarding steps, as highlighted in green in Figure 3. By accelerating these early transformations, Nas1 enables more data products to move efficiently through the pipeline, allowing value to be realized earlier and more consistently.

Currently, the system is capable of handling approximately 20% to 40% of the overall data engineering workload and has reduced over 5,000 hours of manual effort annually. At the same time, we can reduce roughly 2,000 hours of engineering dependency from other stakeholders in the process (e.g. data product managers). Rather than eliminating engineering work, Nas1 shifts focus away from repetitive, low-impact tasks toward higher-value initiatives—such as building complex data transformations, integrating new sources, and supporting advanced client use cases. This reallocation of effort increases scalability and allows Nasdaq to handle more data product requests without expanding headcount, ultimately accelerating delivery timelines.

With these improvements, Nasdaq is positioned to fulfill a significantly larger share of incoming product requests, potentially enabling the onboarding of up to 75% more data products annually. Looking ahead, impact estimates suggest that Nas1 could unlock between \$6 and \$8 million in additional annual revenue by year three, driven by increased engineering throughput

and faster product onboarding—especially as more advanced subagents are integrated into the system.

Strategically, Nas1 empowers business users to define and execute data workflows independently, reducing dependence on engineering resources while improving delivery speed and responsiveness to client needs. Its modular, auditable design provides operational transparency and mitigates the risk of hidden errors, making Nas1 a robust and scalable solution for advancing Nasdaq’s data commercialization strategy.

4 Next Steps and Potential Improvements

To build on the current system and prepare Nas1 for broader deployment within Nasdaq, we have identified several key areas of continued development. These next steps span four strategic dimensions: expanded functionality, system flexibility, and project compatibility.

To improve system flexibility and scalability, we plan to introduce a dedicated prompt engineering agent capable of automatically generating LLM queries from structured product requirements. This will reduce reliance on manual prompt writing when onboarding new agents or extending functionality.

Additionally, we aim to enhance error handling by implementing a fallback mechanism: if code execution fails at runtime, the system can pass the original user query directly to the LLM to regenerate executable code dynamically.

User interaction will also be improved by transitioning from passive input collection to an active, context-aware Q&A interface. This enhancement will be supported by upgrading the current rule-based smart suggestion module to a learned recommendation system that provides more adaptive and personalized guidance during pipeline creation.

Ensuring full project compatibility with Nasdaq’s internal infrastructure is another critical priority. We plan to support local deployment of LLMs to address data privacy and compliance requirements, along with secure API key handling to ensure authenticated access to internal and external services. Additionally, Nas1 will be integrated with internal databases to support real-time data access and streamlined pipeline execution.

Beyond these foundational improvements, several targeted technical milestones are also planned to align the prototype with enterprise deployment standards:

- Integrate live schema indices and metadata from Nasdaq’s internal systems to replace synthetic datasets and validate against real-world production constraints.
- Extend pipeline functionality to support direct commercialization or internal productization of client-facing data outputs.
- Transition language model inference to Nasdaq-hosted infrastructure (e.g., private cloud or on-premise) to meet enterprise-level security, residency, and compliance requirements.
- Expand the agent library to support additional transformation types, advanced validation rules, and domain-specific business logic tailored to Nasdaq use cases.

Collectively, these next steps are expected to enhance the robustness, usability, and integration depth of the Nas1 system—ultimately accelerating time-to-market, increasing engineering leverage, and unlocking greater value from Nasdaq’s data infrastructure.

5 Appendix

5.1 Langgraph Implementation

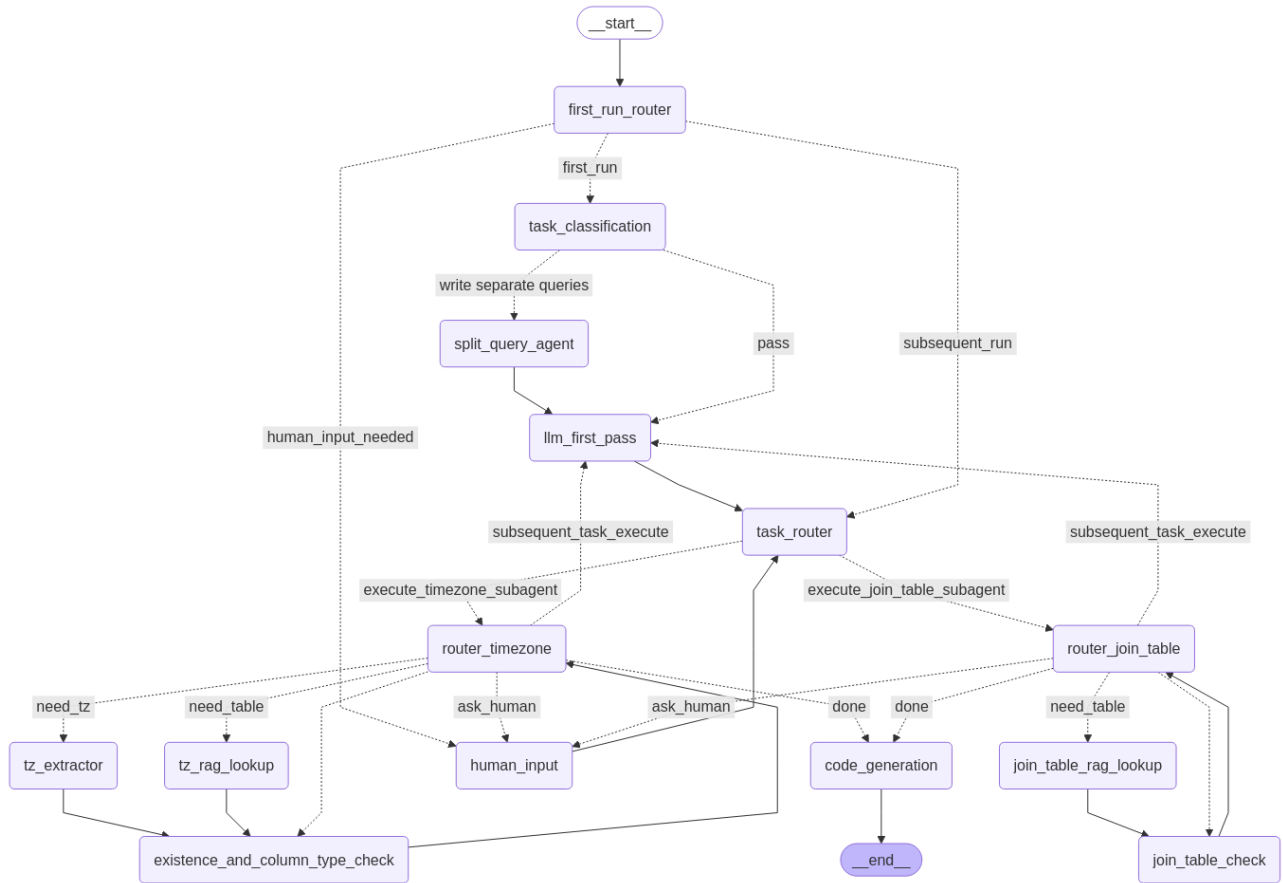


Figure 1: Overview of the Agentic Workflow Architecture Implemented Using the LangGraph Framework [1]

5.2 User Interface

```
NASDAQ GenAI Terminal Connected

System: Connected to server. Ready to process queries.

System: Connected to NASDAQ GenAI Terminal. Type a query to begin.

You: Convert the date column in SKMS table from EST to UTC

System: Processing query...

Detected the following tasks, we will execute it sequentially: ['convert_datetime'].
Additional Info on what each task do: convert_datetime:
+ Convert datetime or timestamps between different timezones or formats.
join_tables:
+ Combine two or more tables together based on matching key columns.

Prompt: Do you want to proceed with these tasks? (yes/no):

Response: yes

Prompt: ? Found invalid inputs: column 'date' cannot be converted to datetime: Column 'date' contains only dates (Year-Month-Day) without time information.
Please provide a column with full datetime values (including hours and minutes)... ⚠ Suggested datetime columns in SKMS: The SKMS table contains the following
datetime column that could be used for timezone conversion: - **New_date**: Datetime each record was processed and uploaded to database (ET - Eastern Time).
Example: 2022-12-20 14:43:32.000 The other date-related column is: - **date**: Date of sale (date type, not datetime), which doesn't include time information.
Example: 2019-01-01 Only the 'New_date' column contains full datetime information (including time) that could be used for timezone conversion. >

Response: Use New_date column

✅ Code ran successfully on attempt 1

✅ Summary of used fields:
```

Figure 2: Example interaction in the NASDAQ GenAI Terminal showing task recognition, user clarification, and execution feedback.

5.3 Nasdaq's Data Deployment Pipeline

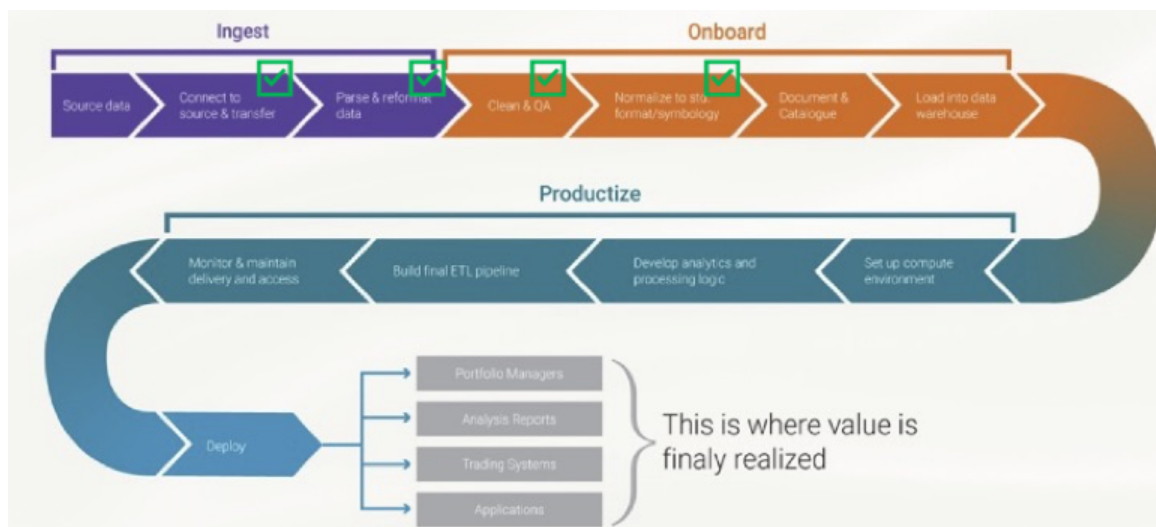


Figure 3: Nasdaq's Data Deployment Pipeline. Green-highlighted steps represent automation opportunities enabled by Nas1.

5.4 Table Schema

Table Name: EFR

Table Description: Point-in-time ranks of public companies' long-term and short-term profitability and growth potentials from five perspectives.

COLUMN	Column DESCRIPTION	TYPE	EXAMPLE	FILTER	PRIMARY KEY
date	Effective date to which the scores apply	Date	2016-01-01	Yes	Yes
name	Company name	String	APPLE INC	No	No
ticker	Company ticker/symbol	String	AAPL	Yes	Yes
cusip	Cusip identifier	String	037833100	Yes	No
eqr	Earnings Quality Rank (scores of 1 for high risk, 5 for low risk, i.e., lowest risk of encountering surprise)	Integer	4	No	No
eqr_expanded	Expanded Earnings Quality Rank (from 1 to 99)	Integer	75	No	No
svr	Strategic Valuation Rank (scores of 1 for overvaluation and elevated risk of share price underperformance, 5 for undervaluation and increased likelihood of share price outperformance)	Integer	5	No	No
svr_expanded	Expanded Strategic Valuation Rank (from 1 to 99)	Integer	90	No	No
amr	Aggregate Momentum Rank (scores of 1 for poor price momentum and elevated risk of share price underperformance, 5 for strong price momentum and increased likelihood of share price outperformance)	Integer	4	No	No
amr_expanded	Expanded Aggregate Momentum Rank (from 1 to 99)	Integer	70	No	No
sgr	Strategic Growth Rank (scores of 1 for poor growth prospects and elevated risk of share price underperformance, 5 for strong growth prospects and increased likelihood of share price outperformance)	Integer	5	No	No
sgr_expanded	Expanded Strategic Growth Rank (from 1 to 99)	Integer	95	No	No
gqr	Growth Quality Rank (scores of 1 for poor earning growth trends, 10 for strong earning growth trends)	Integer	10	No	No
gqr_expanded	Expanded Growth Quality Rank (from 1 to 99)	Integer	92	No	No

Figure 4: Table Schema of EFR table

References

- [1] INC., L. (2024): “LangGraph: Build stateful, multi-agent workflows with LLMs,” GitHub repository, available at <https://github.com/langchain-ai/langgraph>.