ESSEX

EyeAsk

Democratizing Data Through Dialogue Using Generative AI

Anubhav Ranjan	*SVP, Product Development and Deployment*
Pranav Gargieya	*Associate Lead, Development and Deployment*
Akshya Kumar	*Manager*
Kartik Jangir	*Associate Manager*
Jagadeep Vemulapalli	*Senior Analyst*

TABLE OF CONTENTS

1. Introduction: The Data Access Imperative

Modern organizations today collect and store vast amounts of structured data such as sales, customer info, marketing data, customer support, supply chain and much more. Despite the volume and potential value this data presents, most employees within the organization find it difficult to access it readily and easily.

Direct access to data by business users is often restricted or limited. Typically, internal IT / Tech teams act as data owners/stewards and assume responsibility of extracting and sharing data as per user requirements. Even if direct data access may be granted, majority of business users usually lack the technical expertise, metadata knowledge and coding experience to pull relevant data from IT systems.

Business users often rely on reports and dashboards produced by business analytics team to access summarized data information and insights. While these undoubtedly are useful, they offer limited flexibility as they only answer the questions they were initially built to address. For any additional query, enhancements and ad-hoc analysis requests, business users are dependent on data / business analysts.

This lack and inability to access data often results in significant delays, missed business opportunities, impacts customer experience and increased staff expense.

2. How LLMs Can Transform the Data Access Landscape

Large Language Models (LLMs) bring a distinct set of capabilities that make them particularly well-suited for enabling natural language interaction with structured enterprise data. These include:

1. **Understanding Natural Language:** LLMs excel at understanding user intent, even when queries are informal, ambiguous, or conversational - reducing the need for structured inputs or predefined query formats.
2. **Autonomous Code Generation:** LLMs can dynamically generate highly accurate, task-specific code across a range of platforms - SQL, Python etc. - eliminating the need for manual scripting and accelerating complex data tasks.
3. **Context Awareness:** LLMs retain information across turns in a conversation, allowing them to respond in ways that reflect the full context - not just the most recent input - leading to a more coherent and intelligent dialogue.
4. **External Knowledge Integration:** LLMs can draw upon external sources of knowledge—such as business rules, metadata, and domain-specific information—during response generation, enabling outputs that are not only accurate but also context-aware and aligned with organizational standards.
5. **Adaptive Reasoning:** These models can generalize across a wide range of analytical tasks, making them flexible tools for everything from simple data lookups to complex exploratory analysis - with minimal configuration.

Together, these capabilities position LLMs as a transformative foundation for building conversational interfaces - enabling a natural, intuitive engagement between business users and the underlying data infrastructure.

3. Are Out-of-the-Box LLMs Business-Ready?

While Large Language Models (LLMs) are powerful, applying them directly to enterprise data environments without appropriate augmentation often leads to suboptimal outcomes. Generic, off-the-shelf models lack the contextual grounding, structural understanding, and governance safeguards necessary for handling complex, sensitive, and business-specific data use cases. Below are several key limitations that underscore the need for a more robust architecture.

1. **Lack of Contextual Data Understanding**

   LLMs are not trained on, nor do they possess inherent knowledge of an enterprise's data landscape. They lack visibility into data sources, underlying models and structures, field-level definitions, metadata, and the context in which data is used. As a result, their ability to accurately interpret and respond to user queries is significantly constrained. Without proper contextual augmentation, LLMs remain ineffective and unreliable for supporting business-critical decision-making.

2. **Inability to Handle Structured Metadata**

   Off-the-shelf LLMs depend on a prompt-based injection approach, where all contextual information such as data inventory, metadata, calculation logic, and business rules, etc., must be manually included within the prompt as a plain text to guide the model's behaviour and generate relevant responses. There is no efficient way to handle structured metadata natively, making this approach inherently inefficient, fragile, and difficult to scale. It introduces cognitive overload for the model, drives up operational costs due to high token usage, and often results in information loss caused by context window limitations. The consequences of this are incomplete inputs, inconsistent behaviour, and a noticeable decline in response accuracy.

3. **Absence of Built-in Execution Environment**

   While LLMs can produce accurate and relevant code such as SQL or Python, they cannot run that code or interact with enterprise systems on their own. They lack the execution layer needed to connect to databases, process queries and generate output. Even if one manages to pass the data into the prompt, LLMs aren't optimized to scan, filter, and interpret raw tabular data the way a database engine does. They are not designed to process large volumes of structured data in real-time. This creates cognitive overload, leading to degraded performance and inaccurate or incomplete answers.

4. **Poor Multi-Step Workflow Adherence**

   LLMs operate as single, general-purpose agents, without the ability to delegate or coordinate specialized tasks across workflows. Also, as prompts become long and overloaded with context, model performance tends to degrade. Research has highlighted a U-shaped attention bias in LLMs, where the model disproportionately attends to the beginning and end of a prompt, often overlooking the middle- resulting in incomplete or misinterpreted instructions as shown in the figure 3.1. (Liu et al, Lost in the Middle: How Language Models Use Long Contexts. Transactions of the Association for Computational Linguistics-https://doi.org/10.1162/

tacl_a_00638). As a result, single agent LLMs often struggle to reliably and effectively execute complex, multi-step workflows, leading to reduced accuracy and responsiveness.

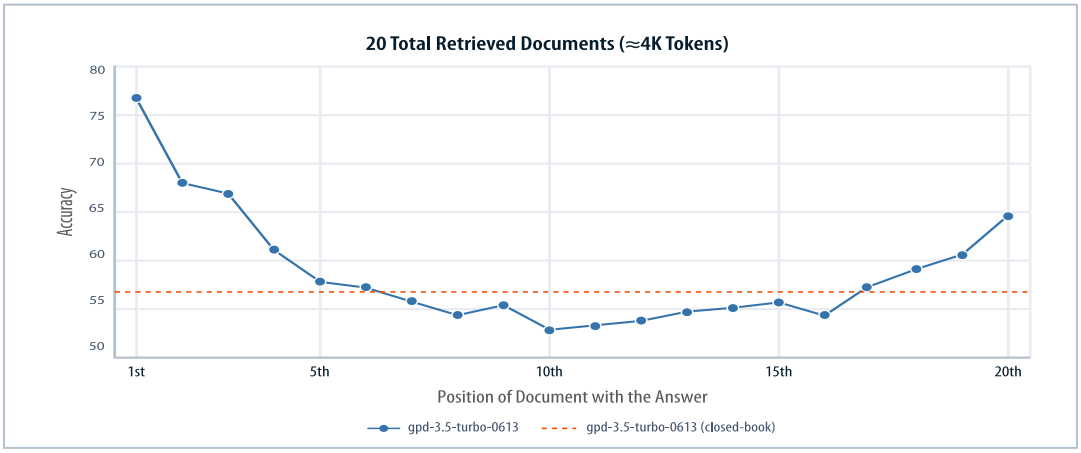

Fig 3.1 - LLM accuracy varies based on the position of relevant information, performing best when answers appear at the beginning or end of the input context.

5. **Potential Data Security, Privacy, and Compliance Risks**

   Off-the-shelf LLMs require access to underlying data to generate responses. In some cases, the data provided to these models may not be immediately discarded and could be temporarily retained before deletion. This raises the risk of exposing sensitive or proprietary information, creating significant concerns around data security, privacy, and regulatory compliance.

4. Key Design Considerations for Building LLM-Based Applications

To address the issues outlined above, this section aims to provide an essential list of design elements to consider while building an accurate, reliable, scalable, and secure LLM based structured data applications.

4.1 Knowledge Base Management Layer

To enable Large Language Models (LLMs) to deliver accurate and contextually relevant responses in enterprise settings—while maintaining strict data privacy and security - a dedicated Knowledge Base Management Layer is essential. This layer serves as a centralized, structured repository that organizes critical metadata and contextual information about enterprise data assets, without requiring direct access to raw data. Typical information captured in the Knowledge Base Management Layer would include (but not limited to):

1. **Data-Level Context:**
   - An inventory of available data tables
   - Descriptions detailing each table's purpose, content, and underlying granularity

2. **Field-Level Metadata:**

   - Field name and its corresponding business name
   - Definition and descriptive annotations
   - List of Unique Values.
   - Common Synonyms
   - Aggregation Logic
   - List of Calculated fields, including their definition, description, formula, aggregation type and synonyms.

3. **Business Context**

   - A business vocabulary mapping commonly used terms to corresponding fields or metrics
   - Reporting definitions (e.g., fiscal year vs. calendar year) to ensure accurate interpretation of time-based queries, especially when organizational reporting periods differ from default LLM assumptions

This structured layer enables LLMs to interpret user queries more accurately by grounding their understanding in real business language, domain-specific rules, and metadata. The richer the metadata and contextual detail provided to the model, the better its performance. As shown in Fig 4.1, LLM response accuracy improves progressively as the knowledge base is enhanced - from basic field names and synonyms to detailed calculation logic and reporting rules.

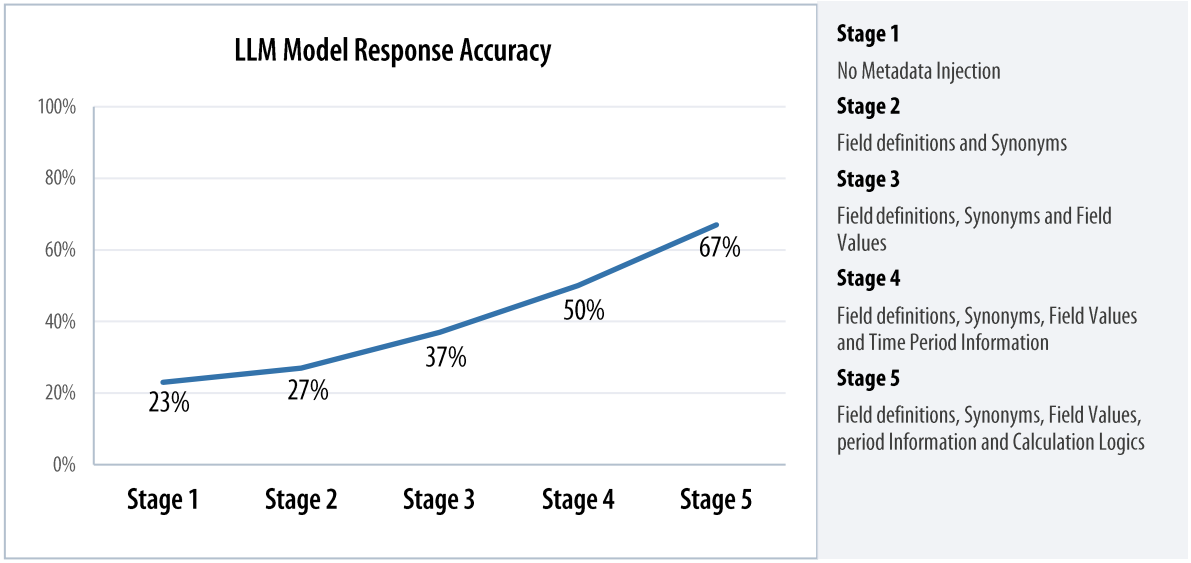

Fig 4.1 - Effect of Additional Knowledge Base on LLM Accuracy.
Testing done on wide format data with knowledge base provided as plain text in prompt.

4.2 Retrieval-Augmented Generation (RAG) Layer

While metadata is crucial for LLM understanding, injecting full metadata into every prompt is often unnecessary, technically limiting, inefficient in terms of model performance. Instead of hardcoding all metadata into the prompt, an enhanced and scalable solution is to provide the LLM with only the 'relevant context' based on the user's query

Retrieval-Augmented Generation (RAG) enables this by dynamically retrieving and injecting just the necessary metadata at runtime, ensuring the model accesses only what it needs to generate accurate and efficient responses. At the heart of the RAG lies embedding-based retrieval. The knowledge base is transformed into dense vector embeddings (numerical representations of text) and stored in a vector database. When a user submits a query, it is also converted into an embedding and compared against the stored vectors. Through similarity search, the system retrieves metadata entries that are semantically aligned with the user's intent. This approach ensures the LLM receives only the most contextually relevant information to generate responses. Figure 4.2 illustrates a standard RAG workflow in action.

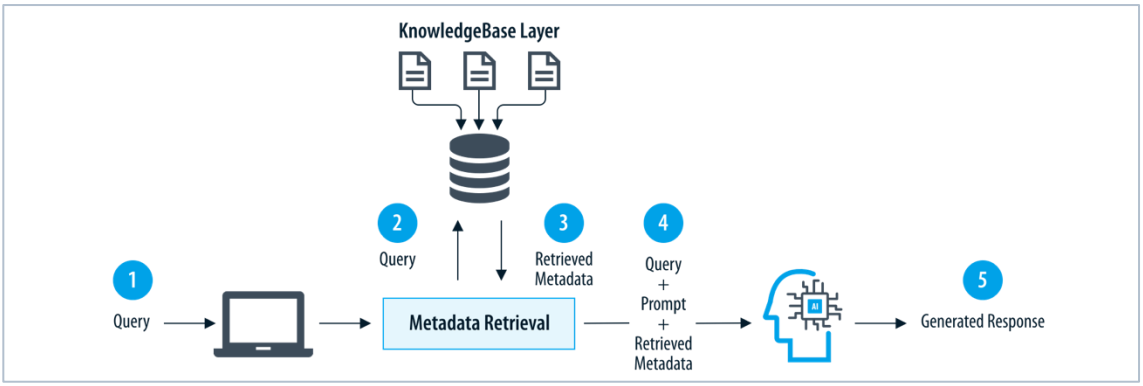

Fig 4.2 - Typical flow of a RAG application

4.3 Metadata Prompt Design

Once the relevant metadata is retrieved from the Knowledge Base, the way it is formatted and included in the prompt can significantly impact the LLM's performance. Because LLMs can only process text inputs, structured metadata - such as tables - must be converted into a readable text format that the model can interpret effectively. Typically, this is done using one of two common approaches:

1. **Unstructured Format**

   In this method, metadata tables are converted into plain text blocks without a fixed format. Fields, descriptions, or rules are embedded directly into the prompt as free-form text. This approach is quick to implement but LLM may misinterpret the structure, importance and relationship between different fields.

   Example:
   "The revenue field refers to total earnings. It is calculated monthly and excludes discounts. Synonyms: income, inflow."

2. Structured Format

This approach preserves the original tabular structure of the metadata by encoding it in machine-readable formats like Markdown tables or JSON blocks.

Example: Markdown Table format

```
| Field Name | Description                          | Synonyms       | Aggregation |
|------------|--------------------------------------|----------------|-------------|
| revenue    | Total earnings excluding discounts | income, inflow | SUM         |
```

Fig 4.3.1 - Structured Metadata in Markdown Table format

Example: JSON format

```
{
  "field_name": "revenue",            // Name of the metric
  "description": "Total earnings excluding discounts",  // Field meaning
  "synonyms": ["income", "inflow"],   // Alternate terms for LLM matching
  "aggregation": "SUM"                 // Aggregation method
}
```

Fig 4.3.2 - Structured Metadata in JSON format

LLMs are more likely to interpret structured formats consistently and accurately than ad hoc text, leading to fewer hallucinations and higher accuracy as shown in Fig 4.3.3.

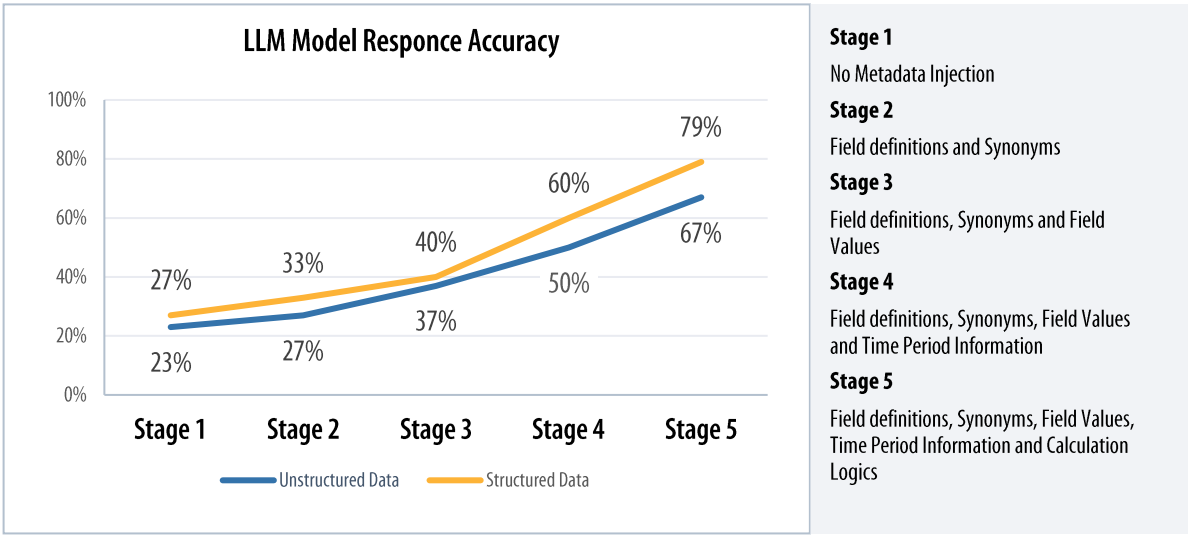

Fig 4.3.3 - Impact of metadata format on LLM accuracy. Testing done on wide format data

4.4 Code Execution Layer

While LLMs can produce logic in languages like SQL or Python, they cannot execute codes, validate output, or enforce access controls. A dedicated code execution layer is essential in LLM-powered applications to safely and reliably bridge the gap between code generation and real-world computation. A separate execution layer enables secure, auditable code execution - ensuring outputs are accurate, compliant, and free from unintended consequences.

Once the LLM agent generates code based on the user's query, it is passed to the Code Execution Layer. This layer performs code validation and safety checks, executes the code securely against connected databases or computation engines, retrieves the results, and forwards them to the agent to respond back to the user.

From an application perspective, this layer improves system reliability, isolates execution risks, and enables robust validation and error handling. At the organizational level, it protects sensitive data, enforces role-based access controls, and supports governance and compliance -making the system secure, scalable, and enterprise-ready.

4.5 Multi-Agent System Architecture

Taking a data related query in natural language and generating a response is a multi-step workflow, comprising a sequence of tasks as illustrated in Fig 4.5 below.

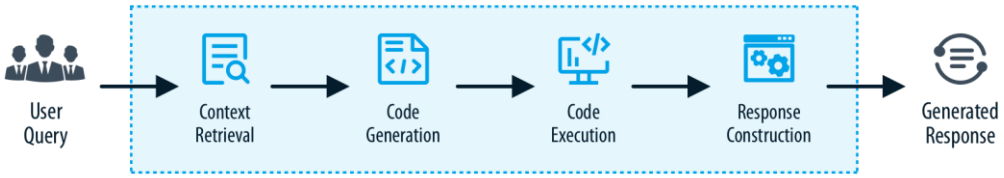

Fig 4.5 - Steps involved in generating a response based on User Query

Given LLMs inability to execute codes, manage complex workflows, and reliably follow multi-step instructions (as outlined in section 3), a multi-agent architecture is critical for building robust, LLM-driven data applications. By decomposing the overall process into distinct, mutually exclusive and collectively exhaustive (MECE) tasks, allows creation of 'specialized agents' responsible for performing a specific, pre-defined role (via individual prompting) within the workflow. This clear division of responsibility ensures that each agent is optimized for its role, can validate and control its own output and exchange only what's needed between agents - minimizing cognitive load and avoiding hallucination issues. This makes the entire system easier to manage and improve, reduces error propagation and improves response consistency.

An important part of this architecture is also **model selection**. Each agent can be assigned the most appropriate model based on its task such as lightweight models (e.g., GPT-4) for straightforward tasks like formatting or summarization, and larger models (e.g., o1, o3) for reasoning-heavy tasks like code generation or query planning. In practice, no single model fits all scenarios. A hybrid model strategy often ensures the best balance between performance, cost, and accuracy - routing high-frequency, low-complexity tasks to fast, economical models, while reserving advanced models for rare, complex operations.

4.6 LLMOps: Observability and Performance Monitoring

For a multi-agent LLM system, it is crucial to track and monitor key performance indicators that reflect how the system performs during real-world use. This includes monitoring both system-wide and specialized agent-level metrics such as response latency, token consumption, agent handoff success, query resolution accuracy, and failure rates. Understanding the behaviour of each agent helps identify inefficiencies, failures, or slowdowns - whether it's a lagging retriever, an underperforming code generator, or a faulty formatter - and allows teams to take focused actions to optimize system performance, reliability and cost.

5. Essex Solution: "EyeAsk"

EyeAsk is a key capability within Essex's Executive Action Response System (EARS) platform, designed to harness the power of Generative AI to simplify data access for business users. It enables users to interact with enterprise data through natural language. EyeAsk interprets business questions in real time and delivers relevant responses in the form of narrative text, visualizations, or both. This allows users to ask questions and receive insights without needing to write complex queries or use traditional BI tools.

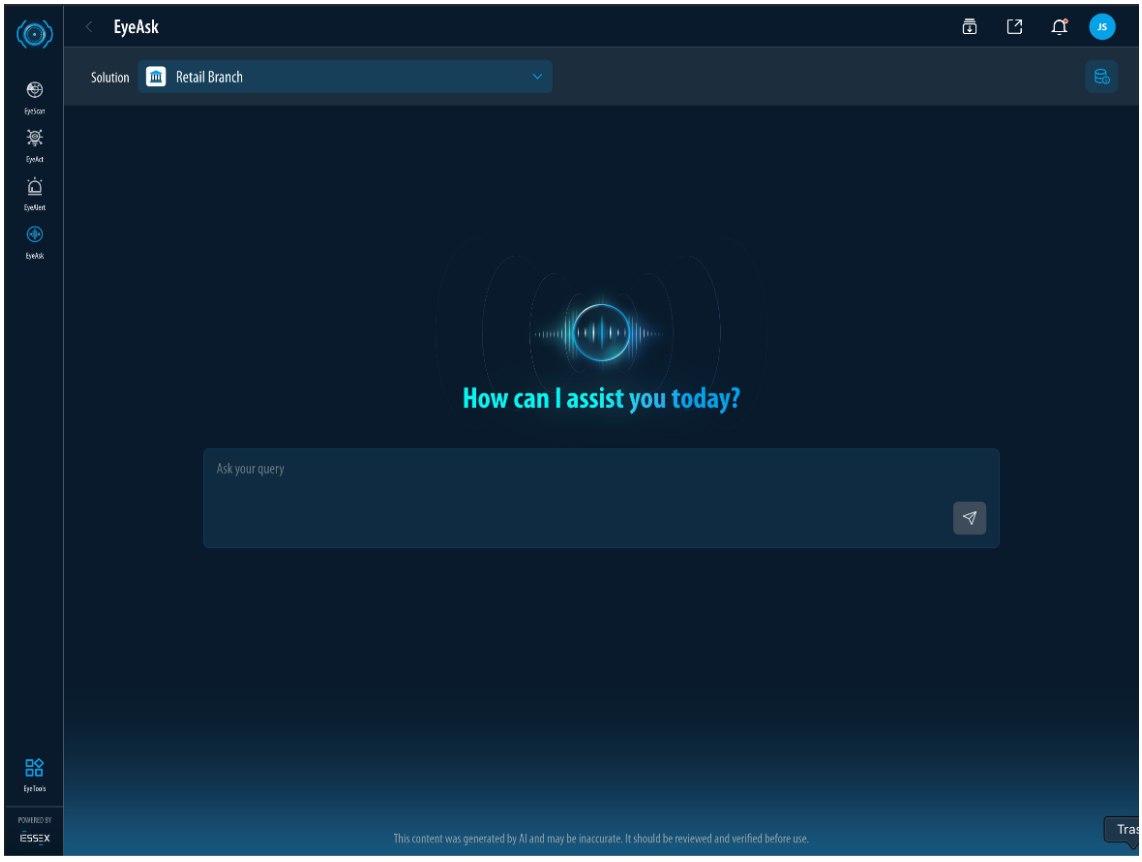

Fig 5.1 - EyeAsk User Interface within EARS

EyeAsk is a compound AI system composed of multiple specialized agents and components, each designed to handle a specific part of the query-to-response workflow. When a user submits a question, EyeAsk first parses and embeds the query, then retrieves relevant table and field information from the Knowledge Base. Leveraging a combination of system prompts, proprietary instructions, and contextual metadata, it translates the natural language input into corresponding Python code. This code is then validated and securely executed on the server, with appropriate access controls enforced. The resulting output is passed to a downstream agent, which generates visualizations and a narrative summary, as applicable, before presenting the final response to the user.

Fig 5.2 depicts the high-level EyeAsk architecture and the query-to-response flow.

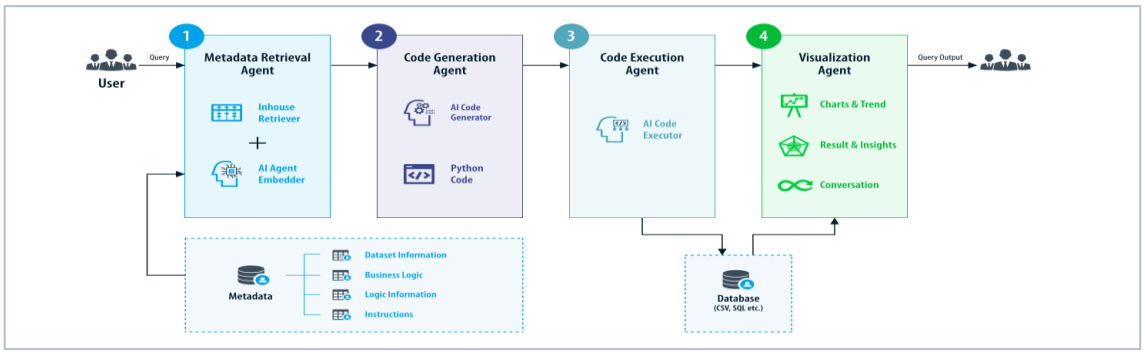

Fig 5.2 – EyeAsk Architecture

6. EyeAsk: Accuracy

6.1 Testing Methodology

To evaluate EyeAsk's accuracy and consistency, we tested its performance across two common data structures: long format and wide format. For each structure, a set of 91 queries was designed to cover a range of business scenarios – including basic and advanced filtering, time-based comparisons, grouping logic, multi-metric scenarios etc. Each query was executed 10 times, resulting in 910 total runs per dataset. The evaluation was fully automated using Python scripts, while each model generated response was manually reviewed and classified as either Pass (correct) or Fail (incorrect).

6.2 Testing Results and Analysis

EyeAsk achieved an accuracy rate of **78.6%** on long format data and **81.8%** on wide format data during testing.

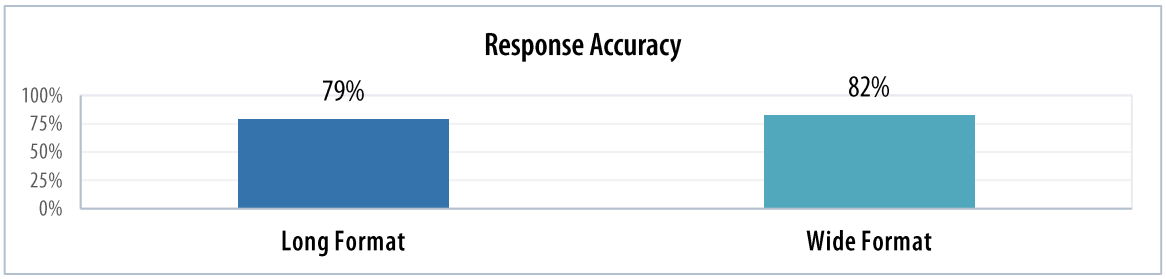

Fig 6.2 - EyeAsk Accuracy by Data Structure

7. Conclusion

EyeAsk directly addresses a persistent challenge faced by modern enterprises: making structured data truly accessible to business users without relying on technical intermediaries. In organizations where data is abundant but locked behind complex systems, technical skill barriers, and dependence on analytics teams, EyeAsk offers a transformational shift. By combining the capabilities of large language models with contextual awareness, modular task orchestration, and secure execution, it enables users to interact with data using natural language.

With an accuracy of ~80%, EyeAsk offers a reliable and scalable alternative to business users to meet their day-to-day data needs. Business users can confidently use the platform to quickly self-serve insights, reduce turnaround times, ease the burden on analytics teams, and accelerate decision-making across the enterprise.

Essex Lake Group LLC

www.essexlg.com