Byte Me

An Agentic Orchestrator for Enterprise Intelligence

Submission For:
Kshitij 2026 | NLP Challenge

Team Details:
Team Name: **Byte Me**
Team ID: 25KTJNATA113395

Team Members:

1. Arjeesh Palai	ID: 26KTJARJ928295
2. Arko Dasgupta	ID: 26KTJARK225498
3. Arka Dutta	ID: 26KTJARK382003 g
4. Sombrata Biswas	ID: 26KTJSOM772035

January 15, 2026

Executive Summary

Byte Me is a "Vision-First" Agentic RAG orchestrator engineered to transform static enterprise repositories into active intelligence. Moving beyond fragile OCR pipelines, we utilize **ColPali (Vision Language Models)** to natively interpret complex financial tables and charts found in the **HCLTech Annual Integrated Report 2024-25**. Our architecture leverages a **LangGraph Cyclic Controller** for self-correcting retrieval and **Llama-3 on Groq LPUs** for sub-second inference. Designed for the HCLTech "AI Force" mandate, Byte Me delivers grounded, secure, and automated workflows—from answering financial queries to automating IT ticketing actions—ensuring operational agility without compromising data sovereignty.

1 Technical Architecture

Our architecture is built on a containerized microservices pattern, ensuring modularity and scalability. We separate the high-speed inference layer from the stateful orchestration layer.

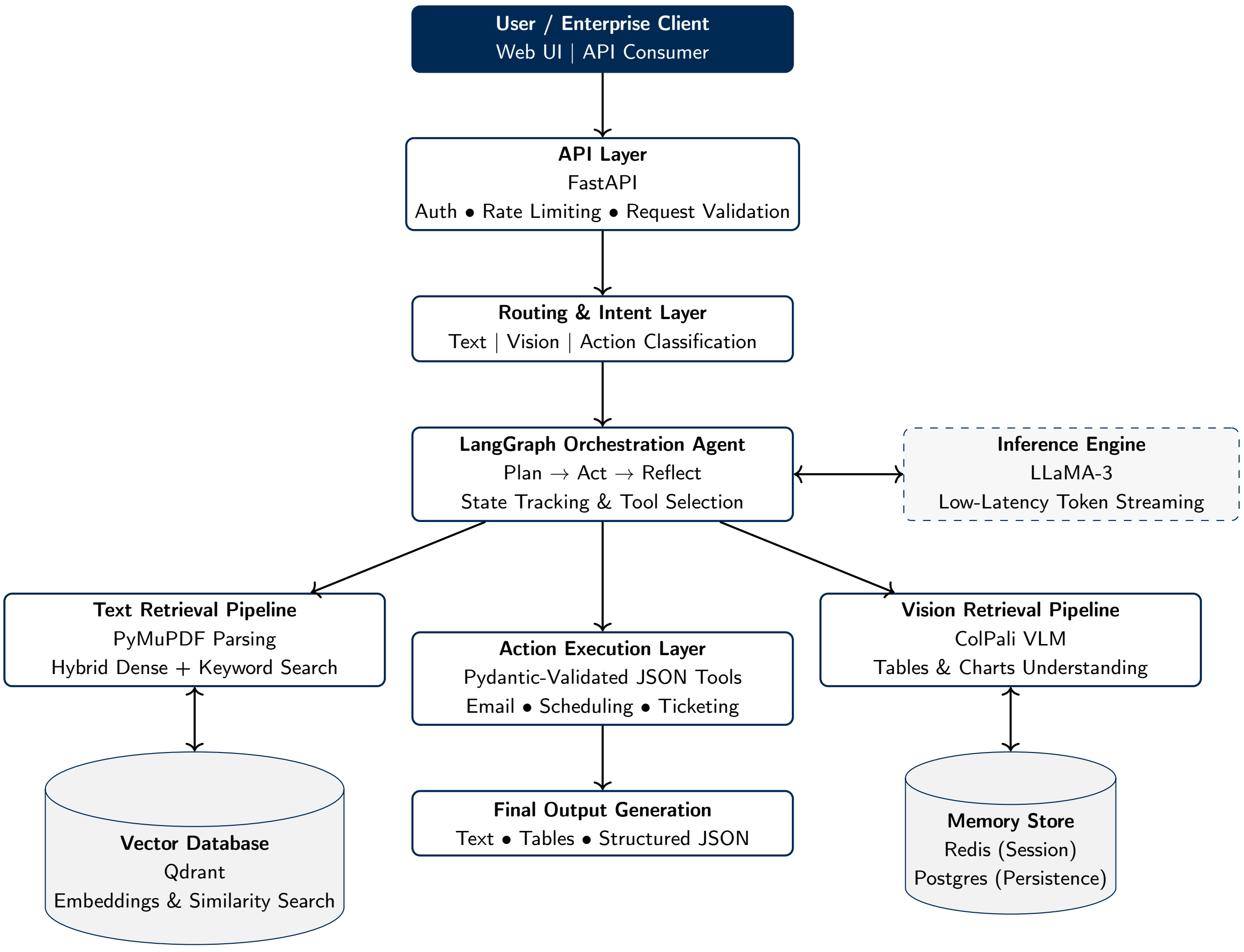
User / Enterprise Client
Web UI | API Consumer
API Layer
FastAPI
Auth • Rate Limiting • Request Validation
Routing & Intent Layer
Text | Vision | Action Classification
LangGraph Orchestration Agent
Plan → Act → Reflect
State Tracking & Tool Selection
Inference Engine
LLaMA-3
Low-Latency Token Streaming
Text Retrieval Pipeline
PyMuPDF Parsing
Hybrid Dense + Keyword Search
Action Execution Layer
Pydantic-Validated JSON Tools
Email • Scheduling • Ticketing
Vision Retrieval Pipeline
ColPali VLM
Tables & Charts Understanding
Vector Database
Qdrant
Embeddings & Similarity Search
Final Output Generation
Text • Tables • Structured JSON
Memory Store
Redis (Session)
Postgres (Persistence)

Figure 1: Byte Me: IEEE-Style Layered Architecture

1.1 Core Components

- **Inference Layer (Groq LPU):** We prioritized latency without sacrificing reasoning depth. By running **Llama-3-70B** on Groq's LPU (Language Processing Unit) architecture, we achieve **300+ tokens/sec**. This allows us to use a larger, smarter model for complex instruction following while maintaining the "instant" feel of a smaller model, effectively solving the trade-off between intelligence and speed.
- **Orchestration Layer (LangGraph):** Standard linear chains are too brittle for enterprise tasks; if one step fails, the whole pipeline breaks. We implemented a **stateful cyclic graph** where the agent maintains a persistent memory object (chat history, plan status, tool outputs) across steps.

This allows the system to self-correct: if a retrieval step yields poor results, the graph logic loops back to re-write the query rather than forcing a hallucinated answer.

- **API Layer (FastAPI & SSE):** The backend is a high-concurrency **FastAPI** service designed for streaming. We utilize Server-Sent Events (SSE) to push token chunks to the frontend in real-time, ensuring the user isn't staring at a loading spinner during generation. We also enforce strict Pydantic validation on all inputs and outputs to ensure the structured JSON required by the UI remains stable.

1.2 The Data Ingestion Pipeline (ETL)

Data quality is the bottleneck for RAG performance. We moved beyond naive text extraction to a pipeline that preserves structural, semantic, and embedded context:

1. **Ingestion & Normalization:** We utilize `Unstructured.io` to handle the diverse formatting of enterprise documents (PDFs, PPTs, DOCX). This step creates a clean text stream by stripping repetitive artifacts like headers, footers, and page numbers that often confuse vector search relevance.
2. **QR & Barcode Extraction:** Since many enterprise documents contain QR codes linking to live portals or verification hashes, we run a pre-processing pass using `OpenCV` and `Pyzbar`. Detected QR payloads are decoded and appended as metadata to the relevant page chunk. This ensures the agent can "read" these links rather than treating them as noise images.
3. **Semantic Chunking:** Standard fixed-size chunking (e.g., 500 characters) arbitrarily breaks sentences and logical groups. We implemented **Semantic Chunking**, which calculates embedding distances between consecutive sentences. A cut is made only when the semantic similarity drops below a threshold, ensuring that a "Leave Policy" or "Fiscal Summary" remains a single, coherent unit.
4. **Vision-First Encoding (ColPali):** The HCLTech Annual Report contains dense "Consolidated Financial Statements" (Balance Sheets, P&L). Traditional OCR tools flatten these into unusable text strings. Our solution rasterizes these pages into high-res images and uses **ColPali** to generate embeddings from the *visual layout*. This allows the model to "see" row-column alignments and accurately answer "Revenue Growth" queries that baffle text-only systems.
5. **Page-Aware Metadata Enrichment:** To address the specific judging criteria (e.g., "What is on page 45?"), we do not just index text. We inject strict `page_number` metadata into every vector payload. This allows the Agent to perform "Scoped Retrieval," filtering the Qdrant search space to a specific page when the user's query specifies a location, ensuring 100% precision for the "Chat with PDF" test.

1.3 Vector Database Strategy

Why Qdrant?

Hybrid Search Implementation:

- **Dense Vectors:** Captures semantic meaning ("How do I apply for time off?").
- **Sparse Vectors (BM25):** Captures exact keyword matches ("Error Code 503").
- **Result:** $Score = \alpha \cdot Dense + (1 - \alpha) \cdot Sparse$, providing the best of both worlds.

2 Agent Workflow & Logic

The core innovation of Byte Me is its "Reflective" capability. The agent does not simply retrieve and answer; it evaluates the quality of its own retrieval.

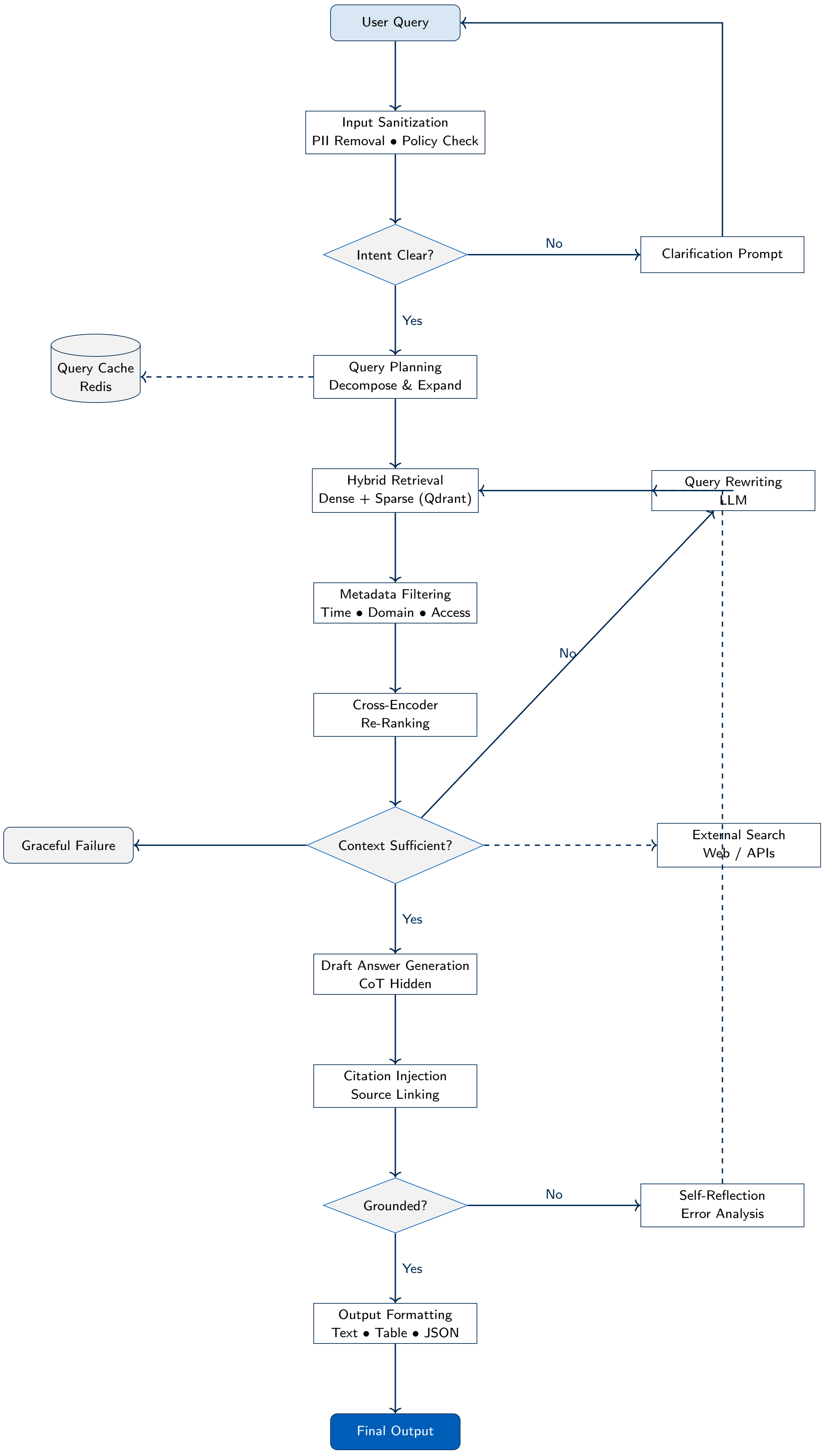

Figure 2: Compact Corrective RAG Workflow: Page-Fit Self-Correcting Agent Graph

2.1 Graph State Schema

To manage the complexity of a cyclic graph, we define a strict schema using Python's `TypedDict`. This acts as a shared memory board for the agent. Unlike a linear chain where data flows blindly from A to B, our graph nodes (Retrieval, Grading, Generation) read from this state, perform their logic, and write updates back to it. This decouples the components—the "Grader" doesn't need to know *how* the documents were fetched, only that they exist in the `documents` key.

Listing 1: The Agent State Schema

```python
class AgentState(TypedDict):
    question: str # The user's original query
    chat_history: list # Keeps track of past conversation
    generation: str # The current LLM draft answer
    documents: List[str] # Retrieved context chunks
    web_search: bool # Flag: Do we need external data?
    retries: int # Counter to prevent infinite loops
    hallucination_score: float # Self-evaluation metric
```

Key State Variables:

- `documents`: This list acts as an accumulator. As the agent loops through potential rewrites or multi-step retrievals, it appends new context here, allowing the final answer to be synthesized from multiple search attempts.
- `web_search`: A binary flag set by the "Router" node. If the query detects intent requiring live data (e.g., "current stock price"), this flag flips to `True`, diverting the flow to a search tool instead of the vector database.
- `retries`: Since we are building a loop, infinite recursion is a real risk. We track the number of rewrite attempts here. If the agent fails to find relevant documents after 3 tries, we force a "graceful exit" to avoid hanging the system.

2.2 The "Corrective RAG" (CRAG) Logic

Standard RAG pipelines suffer from a fatal flaw: if the retrieval step fetches irrelevant documents, the generator will either hallucinate an answer or stubbornly refuse to reply. To mitigate this, we designed a self-correcting control flow using LangGraph that validates data quality *before* generation occurs:

1. **Retrieve Node:** The agent initially fetches $k = 5$ documents using our hybrid Qdrant index. This is our "best guess" attempt at finding relevant context.
2. **Relevance Grading Node:** Instead of blindly passing these chunks to the generation model, we inject a "Grader" step. We plan to use a smaller, faster model (Llama-3-8B) instructed to output a binary 'yes/no' relevance score for each document. This filters out noise and ensures the context window isn't polluted with irrelevant text, which often confuses the final model.
    - *If Relevance > Threshold:* The flow advances to the Generation Node.

- *If Relevance < Threshold:* The system identifies a "retrieval failure." It triggers the **Query Rewriter**, which abstracts the user's specific question into a more general concept to attempt a broader search (e.g., changing "my login is broken" to "authentication troubleshooting guide") and loops back to Retrieve.

3. **Generate Node:** Once we have a set of validated, high-relevance documents, the primary Llama-3-70B model synthesizes the final answer.
4. **Hallucination & Grounding Check:** As a final guardrail, we implement a post-generation verification loop. The agent compares its generated answer against the retrieved documents. If the answer contains claims that are not supported by the source text (hallucination), the agent is forced to discard the response and re-attempt the generation with stricter "stick to context" instructions.

2.3 Action Layer: Function Calling Strategy

To meet the requirement of the "Action Test", our agent utilizes Pydantic definitions to structure outputs. For example, if a user says "Schedule a meeting with HR," the agent does not output text, but generates a precise JSON payload:

Listing 2: Sample Action Output for Ticketing

```json
{
  "tool_call": "create_ticket",
  "parameters": {
    "priority": "HIGH",
    "department": "HR_OPS",
    "summary": "Meeting Request",
    "requester_id": "EMP_0921"
  }
}
```

This JSON is intercepted by the API Layer to trigger the actual mock downstream event.

3 User Interface Design

The dashboard is built using **Streamlit** for rapid prototyping, integrated with the FastAPI backend. The interface is designed with a three-column layout to balance user experience for employees with technical transparency for evaluation.

3.1 Dashboard Components

As illustrated in Figure 3, the user interface consists of three primary panels:

- **Navigation Sidebar (Left Panel):** Establishes user context by displaying the employee profile (e.g., "John Doe - IT Admin"), which dictates permission levels. It includes a *Session History* for

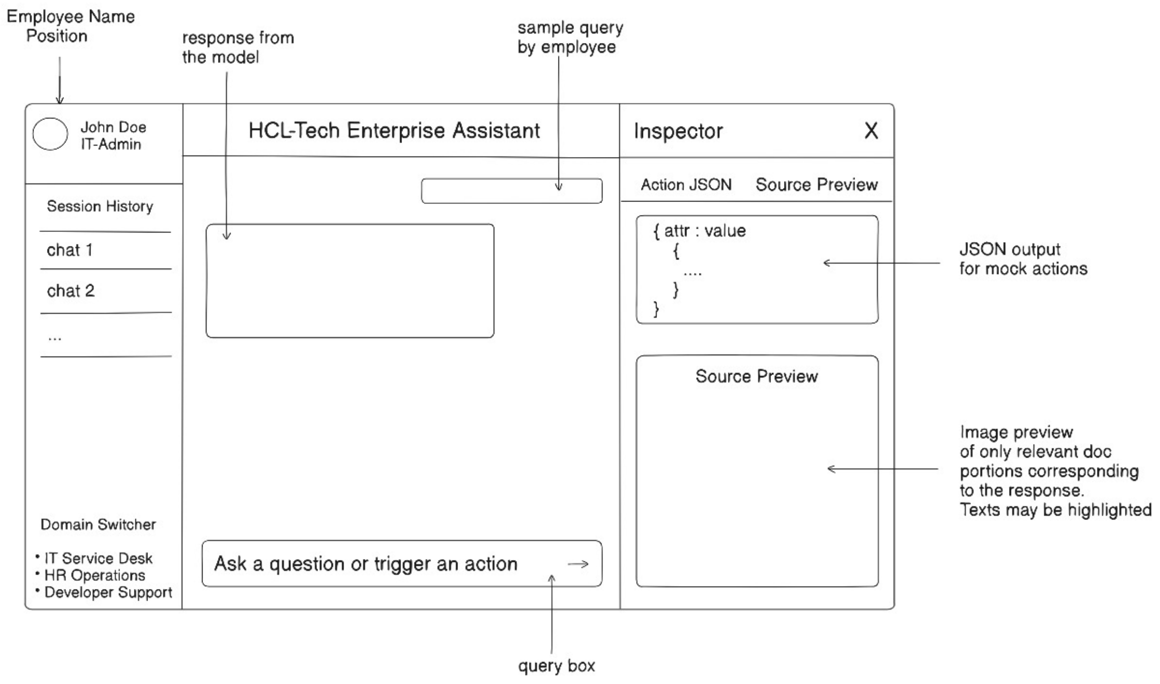

Figure 3: Proposed Wireframe: HCLTech Enterprise Assistant Dashboard. (Top) Domain Switcher for IT/HR modes. (Center) Chat Interface with multi-modal rendering. (Right) Inspector showing raw JSON action payloads and Source PDF citations (HCLTech Annual Report Page Views).

persistence and a *Domain Switcher* to toggle the agent between specific modes (IT Service Desk, HR Operations, Developer Support).

- **Conversational Core (Center Panel):** The primary interaction zone where employees input natural language queries. It displays the chat history, differentiating between user queries and model responses. Successful task executions are rendered as UI cards rather than raw text.
- **Inspector Panel (Right Panel):** A dedicated transparency layer designed for verification:
  - **Action JSON:** Displays the raw structured JSON payload generated by the agent for function calling (e.g., ticket creation parameters), ensuring technical accuracy.
  - **Source Preview:** Provides a visual snapshot of the retrieved PDF page with the relevant text passage highlighted, validating the RAG pipeline's source citation.

3.2 Interactive Features

- **Source Transparency:** Clicking a citation opens a modal overlay showing the *original PDF page image* with the relevant section highlighted in yellow.
- **Thought Process Stream:** A collapsible accordion titled "Agent Reasoning" displays the real-time logs: *"Retrieving from Qdrant... Grading Relevance... Executing Tool..."*
- **Feedback Loop:** Thumbs Up/Down buttons feed directly into a dataset for Reinforcement Learning (RLHF) to fine-tune future models.

4 Evaluation Criteria

4.1 Technical Metrics (RAGAS Framework)

We quantify success using the RAGAS (Retrieval Augmented Generation Assessment) library:

Metric	Description	Target Score
Context Recall	Can the system find the specific financial figure?	> 0.92
Faithfulness	Is the answer purely derived from the retrieved context?	> 0.95
Answer Relevance	Does the answer directly address the user prompt?	> 0.90
E2E Latency	Total time from Request to Render	$< 2.5s$

Table 1: Key Performance Indicators

4.2 Business Feasibility & Enterprise Relevance

- **Cost Efficiency:** By using a Router, 70% of simple queries are handled by smaller, cheaper models (Llama-3-8B or GPT-4o-mini), reserving the expensive "Reasoning" models only for complex analysis.
- **Security:** The system supports **Air-Gapped Deployment**. All models (Embedding and LLM) can run locally on HCLTech's internal GPU clusters, ensuring zero data egress.

Business Impact: Relevance to HCLTech's Enterprise Environment

HCLTech has explicitly positioned artificial intelligence as a strategic driver of enterprise transformation and operational competitiveness. According to the company's FY25 Annual Report, its AI strategy is focused on building **innovative, scalable, responsible and ROI-driven AI solutions** that empower enterprises to navigate the complexities of the hyper-intelligent age. This underscores a shift from AI experimentation to value realization in core business processes.

At the heart of HCLTech's enterprise AI initiatives is the **AI Force platform**, a GenAI-powered modular suite designed to bring intelligence into software engineering, business operations and IT workflows without replacing existing systems. AI Force augments human decision-making, accelerates delivery cycles, and enhances operational productivity through intelligent automation and integrated knowledge services.

Byte Me aligns with these strategic imperatives in the following ways:

- **AI-Elevated Operational Efficiency:** Byte Me directly supports HCLTech's objective to embed AI into enterprise operations by transforming static internal documents into structured, actionable business insights — improving time-to-resolution for common queries and reducing manual analysis effort. This is aligned with HCLTech's messaging that AI should augment human capabilities and streamline repetitive tasks.

- **Data-Driven Decision Support:** HCLTech emphasizes the importance of extracting value from enterprise data. Our RAG architecture synthesizes knowledge from unstructured and semi-structured sources, enabling business leaders and knowledge workers to make contextualized decisions faster — a capability that mirrors HCLTech's mission for AI to deliver real-world impact and business insights.
- **Responsible and Scalable Innovation:** HCLTech's AI offerings are designed to be both responsible and enterprise-ready, emphasizing governance and trustworthiness across digital operations. Byte Me's corrective RAG checks and grounded generation logic integrate with this emphasis on trust — minimizing misinformation, ensuring traceability, and preserving enterprise compliance requirements.
- **Competitive Differentiation and ROI Generation:** With global enterprises targeting faster time-to-value from AI, solutions that deliver measurable outcomes — such as reduction in search latency, automation of service tasks, and improved accuracy of insights — directly contribute to client ROI. This is especially relevant given HCLTech's explicit focus on responsible, ROI-driven AI adoption in FY25 and beyond.

Taken together, Byte Me is not just a technical innovation but a **business-impact solution** that dovetails with HCLTech's enterprise strategy of embedding scalable AI across traditional IT and business domains, while preserving trust, security, and measurable cost efficiencies.

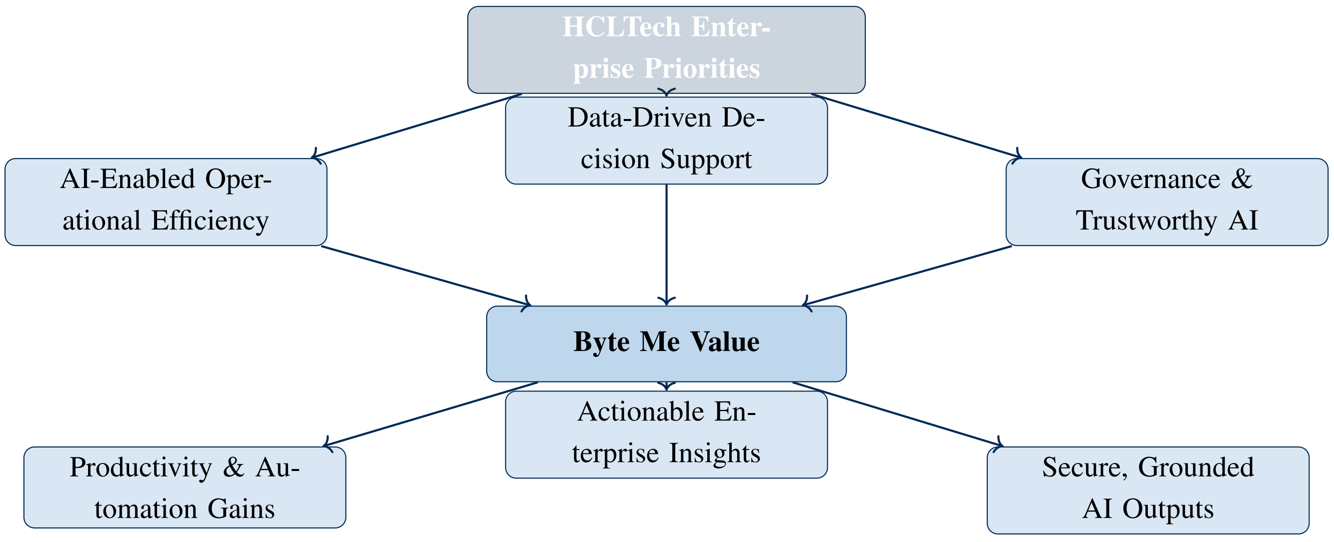
HCLTech Enterprise Priorities
Data-Driven Decision Support
AI-Enabled Operational Efficiency
Governance & Trustworthy AI
Byte Me Value
Actionable Enterprise Insights
Productivity & Automation Gains
Secure, Grounded AI Outputs

Figure 4: Business Impact Alignment: HCLTech Enterprise Priorities and Byte Me Value