Technical Report: Multi-LLM AI Support Automation via OpenRouter

Dr. Paulo Ricardo Protachevicz

Abstract

This report presents an advanced solution for customer service automation, combining multiple language models (LLMs) via OpenRouter and a vectorized corporate knowledge base. The goal is to ensure cost reduction, fast response, and accuracy through intelligent routing, semantic search, and integration with existing Xano/Postgres stack.

1. Architecture and Service Flow

The architecture consists of four main layers: intelligent LLM router, vectorized knowledge base, multi-channel integration module, and a monitoring layer.

Detailed flow:

1. **Reception:** The user interacts via chat, Slack, email, or API.
2. **Semantic Search:** The system queries the vector database (Postgres/pgvector) using embeddings to find similar answers; if a relevant match exists, the answer is returned instantly.
3. **Multi-LLM Routing:** If there is no direct answer, the router evaluates the complexity of the query (word count, technical keywords, history) and selects the most appropriate LLM via OpenRouter—opting for economical models for trivial questions and premium models for critical cases.
4. **RAG and Contextualization:** If the vector search returns related excerpts, these are provided to the LLM via RAG (Retrieval-Augmented Generation), increasing accuracy and contextualization.
5. **Logging:** Every interaction (question, answer, context, model) is logged and stored, enriching the vector database for future searches.

Practical example: A user asks about "delivery time." The system finds a similar answer in the vector database (FAQ) and returns it instantly. For new or complex questions, the router directs the query to the appropriate LLM, inserting corporate data into the prompt and generating a personalized answer.

2. Implementation and Integration

The prototype was developed in Python, using modular functions for routing, LLM calls, vector search, and logging. Postgres/pgvector enables similarity queries (`cosine`, `L2`) via SQL, simplifying the architecture while maintaining performance.

Automated data ingestion: Connectors (e.g., n8n) capture information from Slack (support chats), email (resolved cases), and Google Drive (manuals and policies), converting relevant content into embeddings and enriching the knowledge vector. The ingestion routine can be scheduled (e.g., daily indexing of new files), keeping the base always updated and relevant.

Integration: The system operates as a microservice (FastAPI), which can be called by the backend via HTTP API, facilitating integration and maintenance.

3. Benefits and Efficiency

Main advantages:

- Reduction of operational costs (30–80% token savings, according to benchmarks).
- Fast and consistent responses, leveraging history and institutional knowledge.
- Continuous improvement: new interactions enrich the vector, supporting incremental learning.
- Flexibility to integrate new models and data sources.
- Compliance with LGPD, enabling anonymization and control of sensitive data.

Efficiency in practice: Frequently asked or previously answered questions are handled almost instantly, while new or complex queries receive special attention with accuracy and supporting evidence. The learning cycle is closed by recording each new case in the vector database.

4. Technical Challenges and Recommendations

Current challenges:

- **Efficient routing:** Requires calibration and may evolve into ML-based intent classifiers.
- **Latency:** Vector search and external LLM calls may increase response time; optimizing cache and indexing is essential.
- **Consistency and security:** Standardize the response tone across models and ensure anonymization of PII before sending data to external APIs.
- **Infrastructure:** Enabling pgvector may require IT support; consider alternatives like Supabase, Pinecone, or Qdrant if needed.

Recommendations for evolution:

1. Continuous refinement of routing rules, collecting metrics and user feedback.
2. Expand data ingestion sources (Slack, email, Drive) and maintain automated update routines.
3. Implement monitoring dashboards (cost, usage, quality, fallback).
4. Document security, failover, rate limiting, and data handling procedures.

Conclusion

The proposed architecture enables a new level of automated support, balancing cost, efficiency, continuous learning, and security. The model is flexible and can evolve as demand grows and new requirements emerge, positioning the organization at the forefront of practical conversational AI in support operations.