A Multi-Agent LLM Architecture with Dynamic Cascading for On-Premises, LGPD-Compliant Organizational Assistants

Renato de Oliveira Rosa[1]

[1]Fucape Business School – Espírito Santo – ES – Brazil

gestor.renatorosa@gmail.com

Abstract. *The integration of large language models (LLMs) into data-sensitive organizational environments faces two structural barriers: regulatory constraints (in Brazil, the LGPD) and workload heterogeneity. This paper presents a multi-agent LLM architecture combining full on-premises execution with a dynamic cascading router: a heuristic router, four specialized LLM agents (Tier 1 ultra-light to Tier 3), a per-workspace RAG agent, a compliance and audit agent, and a real-time performance agent. The router dispatches each query to the cheapest viable agent, cascading to lighter tiers on failure. We describe the architecture, the LGPD masking pipeline, prompt-injection mitigations, and one integrated demonstration, released as open source.*

Resumo. *A integração de modelos de linguagem de grande porte (LLMs) em ambientes organizacionais que tratam dados sensíveis enfrenta duas barreiras: restrições regulatórias (no Brasil, a LGPD) e a heterogeneidade das cargas de trabalho. Este trabalho apresenta uma arquitetura multi-agente de LLMs que combina execução totalmente local com um roteador dinâmico em cascata: um roteador heurístico, quatro agentes LLM especializados (Tier 1 a Tier 3), um agente de RAG isolado por workspace, um de auditoria e um de desempenho. O roteador despacha cada consulta ao agente mais barato capaz de respondê-la, com fallback transparente em caso de falha, e o sistema foi publicado como código aberto.*

1. Introduction

The incorporation of LLMs into organizations that process personal data is limited by more than raw performance. Data-protection regulations such as the Brazilian LGPD [Brasil 2018], the European GDPR [da União Europeia 2016], and the California CCPA [da Califórnia 2018] impose strict requirements on data processing, which in most practical scenarios makes it unfeasible to send queries and documents to remote LLM providers, creating vendor lock-in and data-sovereignty bottlenecks. At the same time, the heterogeneity of desktop workloads, from trivial greetings to legal analysis and code generation, makes a single large model for every query economically unfavorable.

This paper presents a multi-agent LLM architecture executed entirely on-premises and designed to serve as a corporate assistant that reconciles data sovereignty, regulatory compliance, and computational efficiency. The solution articulates a society of cooperating agents around a *dynamic cascading router*: a heuristic router agent, four specialized LLM agents, a per-workspace RAG agent, a compliance and audit agent, and a real-time performance agent. The technical contributions are concrete and observable: (i) an

explicit, thread-safe router whose decision (tier, model, fallback chain) is exposed in real time through a routing pill in the frontend; (ii) a cascading protocol that converts model failures into transparent fallbacks; (iii) an audit pipeline that masks personal data *inside* the log itself, keeping the audit repository LGPD-compliant; and (iv) per-workspace vector isolation treated as a security boundary. This camera-ready version addresses the review feedback, covering the page limit, Related Work against concrete systems, prompt-injection mitigations, verbatim LGPD regex and system prompt, clarified harness ground truth, and a single integrated demonstration. The complete source code is public at `https://github.com/renato0503/A-Multi-Agent-LLM-Architecture-with-Dynamic-Cascading-for-On-Premises` [Rosa 2026].

2. Related Work

Three research lines ground the proposal: multi-agent systems, LLM agents, and model cascading. This section compares the architecture with concrete related systems rather than restating textbook concepts.

Multi-agent systems. Classic frameworks such as the BDI model [Rao and Georgeff 1995] and the Jason language [Bordini et al. 2007] formalize autonomous agents that perceive and act on their environment [Wooldridge 2009], providing the conceptual repertoire to describe systems in which specialized components cooperate toward a common goal. In this architecture, each tier is modeled as a specialized agent with that property.

LLM agents. Chain-of-Thought [Wei et al. 2022] and ReAct [Yao et al. 2023] improve reasoning by eliciting intermediate steps and combining reasoning with action; Toolformer [Schick et al. 2023] and Generative Agents [Park et al. 2023] extend this line by letting agents invoke APIs and interact in simulated environments. In this work, each LLM tier is an agent that answers a subset of queries under a router, and, unlike tool-using agents, it performs no external tool invocation, which constrains the attack surface.

Model cascading. CALM [Schuster et al. 2021] slows decoding at critical parts of the text; FrugalGPT [Chen et al. 2023] orchestrates chains of models of varying size to cut costs; Hybrid LLM [Ding et al. 2024] routes queries by estimated complexity. These approaches treat routing as a classification problem. The present work shares that principle but implements it as an explicit, observable, thread-safe router agent whose behavior is exposed to the user and whose failure handling is a first-class cascading protocol (Section 3.2).

RAG. Retrieval-augmented generation [Lewis et al. 2020, Gao et al. 2023] grounds answers in verifiable external sources. The architecture incorporates a RAG agent whose vector base is isolated per workspace (Section 3.3).

Local inference and compliance. On-premises inference with open models served by Ollama [Ollama 2024] and orchestrated by LangChain [LangChain 2024] is viable for edge deployments, and the extraction of training data from models is a known risk [Carlini et al. 2021]. This work sits at the intersection of these agendas: fully local operation plus a masking pipeline that keeps both the models and the audit trail compliant with the LGPD.

Table 1 compares representative systems across the dimensions that distinguish the proposal: observable routing, cascading fallback, on-premises operation, LGPD masking, and per-tenant isolation.

Table 1. Comparison of related systems and the proposed architecture.

System	Routing	Fallback	On-prem	LGPD mask	Isolation
FrugalGPT [Chen et al. 2023]	cost-based	cascade	—	—	—
Hybrid LLM [Ding et al. 2024]	learned	—	—	—	—
CALM [Schuster et al. 2021]	token-level	—	—	—	—
ReAct/Toolformer [Yao et al. 2023, Schick et al. 2023]	tool-based	—	—	—	—
This work	heuristic, visible	cascade	yes	in audit log	per workspace

3. Multi-Agent Architecture

The architecture is a society of cooperating agents, each implemented in a dedicated module. Figure 1 shows the overall view: the router agent dispatches each query to one of four specialized LLM agents, while the RAG, audit, and performance agents observe all interactions.

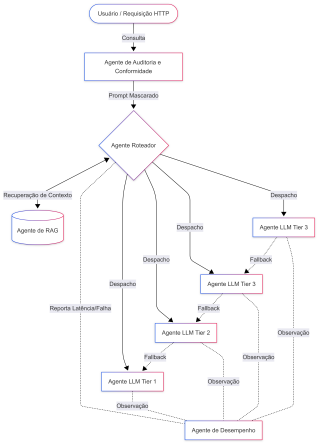

Figure 1. Overview of the multi-agent architecture.

3.1. Router agent

The router agent (`llm_router.py`) exposes `classify(prompt, rag_context, requested_model)`, returning a `RoutingDecision` with the selected tier, a human-readable justification, the context size, and the fallback chain. The decision is built in stages: (1) an explicit user override is respected; (2) a compiled regex detects complexity signals (code: python, regex, sql, api; calculation: calcule, derivada; tax/fiscal: ICMS, ISS, SPED, simples nacional; legal: lei, artigo,

jurisprudência, análise crítica); (3) the prompt+RAG context is compared against thresholds: above 6,000 characters routes to Tier 3, above 4,000 to Tier 2, otherwise Tier 1; (4) small-talk and greetings prevail over context size and are always answered by Tier 1. The effective Tier 1 model is chosen adaptively from the models installed in Ollama, ordered from lightest to heaviest. Router state and model invocation are protected by locks to avoid GPU/CPU contention.

3.2. Cascading protocol with fallback

The cascading protocol is implemented in `call_with_fallback`. For each tier in the chain, `_try_model` records the latency, the HTTP code returned by Ollama, and either the generated text or a structured error. The first tier that returns a non-empty response wins. The canonical order is `[TIER_3_CODE, TIER_3_REASON, TIER_2, TIER_1]`, trying the most capable model first, so that a failed agent is observed by the orchestrator and handled through cooperation rather than surfaced to the user. In practice, the models are `qwen2.5:1.5b` (1.5 B, Tier 1), `gemma3:latest` (4.3 B, Tier 2), `gemma4:latest` (8 B, Tier 3-Reasoning), and `llama3.1:8b` (8 B, Tier 3-Code), all served locally by Ollama.

3.3. RAG agent with workspace isolation

The RAG agent (`workspaces.py`) materializes each workspace as a ChromaDB collection under `./workspaces_db/{slug}/`. It ingests PDFs via `PyPDFLoader` and `RecursiveCharacterTextSplitter` (blocks of $\sim$1,000 characters, 200-character overlap), generates embeddings with `nomic-embed-text`, and supports per-document deletion, honoring the LGPD right to be forgotten with physical removal of the corresponding vectors. Additionally, workspace isolation is treated as a security requirement: a query cannot retrieve documents from a different knowledge domain.

3.4. Audit and compliance agent

The audit agent (`audit.py`) is a Starlette middleware that intercepts every HTTP request. Before dispatch, the JSON body is serialized and passed to `mask_sensitive_data`, which applies regular expressions for CPF, CNPJ, e-mail, phone, and credit-card numbers, and writes a structured line to `audit.log` (UTC timestamp, source IP, method, path, status, masked body, user id). Because the log stores no personal data in clear text, the audit repository itself remains LGPD-compliant, resolving what we call the *audit paradox*: enabling full traceability of the system's actions while ensuring the audit artifact does not become a liability of personal-data leaks.

3.5. Performance agent and sovereign offline mode

The performance agent (`performance.py`) is a thread-safe in-memory collector exposing `GET /api/metrics` (P50/P95/P99 latency percentiles per route and a histogram of served models over the last five minutes), `GET /api/metrics/stream` (Server-Sent Events consumed by the `/metrics` dashboard), and `POST /api/metrics/clear` (admin only). This makes the router and the cascade observable in real time. The `CERRA_REMOTE_MODE` flag switches persistence and authentication backends behind the same `firestore_client` interface: by default, persistence runs on a local SQLite file and authentication is implicit (local user, admin role); in remote mode, the same interface delegates authentication to Firebase and persistence to Firestore, keeping a local-network-token fallback.

4. Security and Compliance

This section presents the LGPD masking pipeline with verbatim regex patterns and discusses prompt injection, a concern raised by the reviewers.

4.1. LGPD masking pipeline

The masking pipeline lives in `config.py` and `audit.py`. The patterns below are reproduced verbatim from the source code:

```
LGPD_SENSITIVE_PATTERNS = [
    r"\b\d{3}\.?\d{3}\.?\d{3}-?\d{2}\b",                    # CPF
    r"\b\d{2}\.?\d{3}\.?\d{3}/?0001-?\d{2}\b",              # CNPJ
    r"\b\d{4}[- ]?\d{4}[- ]?\d{4}[- ]?\d{4}\b",             # credit card
    r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b", # e-mail
    r"\b\d{4,5}-?\d{4}\b",                                  # phone
]
LGPD_MASK = "[REDACTED]"
```

Every audit line is stored with any occurrence of these patterns replaced by `[REDACTED]`, so that the audit trail never persists personal data in clear text, even when a request body legitimately contains it.

4.2. Prompt injection and mitigations

LLM-integrated applications are exposed to direct and indirect prompt injection, in which adversarial instructions are smuggled into the prompt or into retrieved content [Greshake et al. 2023]. The architecture applies defense in depth:

- **No external tool invocation.** Unlike tool-using agents [Schick et al. 2023, Yao et al. 2023], the LLM agents call no external functions or network endpoints; their only input is the grounded context retrieved by the RAG agent.
- **Workspace isolation.** Retrieved content is bounded to a single workspace, preventing an injected document from influencing queries in another domain (Section 3.3).
- **Grounding directive in the system prompt.** The system prompt (Section 6.3) instructs the model to answer factual queries strictly from the provided context and to refuse when the information is absent.
- **Full audit trail and bounded exposure.** Every request and response, with personal data masked, is recorded, enabling forensic review; in remote mode, access is gated by a local-network token or Firebase authentication.

These measures do not eliminate injection entirely; defense against adversarial content remains open work, as discussed in Section 7.

5. Integrated Demonstration Scenario

To address the review feedback, the four original scenarios were consolidated into a single, continuous demonstration on the `politicas-rh` workspace, which exercises every distinguishing feature of the system in sequence:

1. **Setup.** The presenter creates the `politicas-rh` workspace and uploads an internal human-resources regulation (PDF), indexed into an isolated ChromaDB collection.

2. **Progressive tier escalation.** Three queries of increasing complexity are asked: (a) *"Olá"* is routed to Tier 1 (`qwen2.5:1.5b`); (b) *"Quantos dias de férias a cláusula 12 prevê?"* triggers Tier 2 (`gemma3`) grounded on the retrieved clause; (c) *"Compare as cláusulas 12 e 15 e faça uma análise crítica"* routes to Tier 3-Reasoning (`gemma4`). The routing pill updates after each answer.
3. **Override and failure injection.** The presenter forces Tier 3-Code (`llama3.1:8b`) via the gear menu and asks for a Python function to validate a fiscal identifier; the performance panel records the increased latency. The presenter then terminates the underlying process; the next lighter tier answers the following query transparently, and the attempt list is recorded by the performance agent.
4. **Masking inspection and right to be forgotten.** A request containing a valid-format CPF, e-mail, and phone is submitted; the corresponding `audit.log` line and the `/admin/audit` view show the values as `[REDACTED]`. The presenter then deletes one document via `DELETE /admin/workspace/{slug}/document/{filename}` and shows the corresponding vector blocks physically removed from ChromaDB.
5. **Workspace as security boundary.** A plausible query from a second workspace (`seguranca-ti`) is submitted while `politicas-rh` is active; the system correctly reports the absence of information, and the same query in the correct workspace produces a grounded answer citing the retrieved block.

Figure 2. Frontend chat with the routing pill (top right) showing the active tier and latency in real time.

The scenario is reproducible from a clean installation with the four cascade models downloaded in Ollama, as documented in the repository [Rosa 2026]. A narrated video walkthrough of the same scenario accompanies the paper.

6. Empirical Observations and Reproducibility

6.1. Routing distribution and latency

Observations from productive use reveal a workload dominated by low-complexity queries: approximately 70% are answered by Tier 1, 20% by Tier 2, and 10% by Tiers 3, consistent with the cascading literature [Chen et al. 2023, Ding et al. 2024]. Consequently, resolving 70% of demand with a 1.5 B model frees GPU resources for complex reasoning and code. In a 30-day window, fewer than 1% of queries required fallback, almost all due to out-of-memory errors of Tier 3-Reasoning on long contexts; without the cascade, those failures would have surfaced as user-visible errors.

6.2. Evaluation harness and ground-truth construction

Future work includes a shared evaluation harness built over a *frozen, fully synthetic snapshot* of the demonstration workspaces, in which (i) routing accuracy is measured against *expert-annotated expected tiers* for generated queries, (ii) masking recall and precision are computed by injecting *known synthetic PII* (fabricated CPF, CNPJ, e-mail, phone, and card numbers) into synthetic documents, and (iii) fallback frequency and latency percentiles are measured on *aggregated, de-identified production logs*, avoiding re-identification risk and the ambiguity of unlabeled ground truth.

6.3. Reproducibility

The complete source code is public at `https://github.com/renato0503/A-Multi-Agent-LLM-Architecture-with-Dynamic-Cascading-for-On-Premises` [Rosa 2026]. The system prompt reproduced below is the actual prompt used by the production backend:

```
"Você é o Cerra.AI, um assistente corporativo inteligente, seguro e de tom
profissional e humano.

Diretrizes:
1. Para saudações, interações sociais básicas e perguntas sobre o histórico
   da própria conversa corrente, responda de forma natural, educada e prestativa.
2. Para perguntas factuais ou consultas sobre normas, documentos e dados da
   empresa, use estritamente o Contexto fornecido abaixo. Se as informações
   necessárias para responder a essas consultas não estiverem no Contexto,
   informe polidamente que não possui essas informações nos documentos fornecidos.

Histórico da Conversa:
{chat_history}

Contexto:
{context}"
```

Together with the regex patterns of Section 3.4 and the routing heuristic of Section 3.1, this allows third parties to reproduce the router decisions, the masking behavior, and the fallback protocol exactly.

7. Conclusion and Future Work

This paper presented a multi-agent LLM architecture executed entirely on-premises and organized around a dynamic cascading router, combining a heuristic router agent, four specialized LLM agents, an isolated RAG agent, an audit agent with LGPD masking, and a real-time performance agent. The system is in production use and released as open source for reproducibility.

Future work includes replacing the heuristic router with a learned classifier trained on the audit and performance flow, a negotiation protocol in which low-confidence signals trigger explicit escalation, publishing the evaluation harness of Section 6.3, and output-side guardrails and input sanitization to further harden prompt-injection defenses.

References

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). *Programming Multi-Agent Systems in AgentSpeak using Jason*. John Wiley & Sons.

Brasil (2018). Lei n. 13.709, de 14 de agosto de 2018: Lei geral de proteção de dados pessoais (lgpd). `https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/l13709.htm`.

Carlini, N., Ippolito, D., Jagielski, M., et al. (2021). Extracting training data from large language models. In *Proceedings of the 30th USENIX Security Symposium*, pages 2633–2650. USENIX Association.

Chen, L., Zaharia, M., and Zou, J. (2023). Frugalgpt: How to use large language models while reducing cost and improving performance. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3299–3313.

da Califórnia, E. (2018). California consumer privacy act (ccpa). `https://leginfo.legislature.ca.gov`.

da União Europeia, C. (2016). Regulamento (ue) 2016/679 relativo à proteção das pessoas singulares (gdpr). `https://eur-lex.europa.eu/eli/reg/2016/679/oj`.

Ding, D., Mallick, A., Wang, C., et al. (2024). Hybrid llm: Cost-efficient and quality-aware query routing. In *Proceedings of the 12th International Conference on Learning Representations*. OpenReview.

Gao, Y., Xiong, Y., Gao, X., et al. (2023). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Greshake, K., Abdelnabi, S., Mishra, S., et al. (2023). Not what you've signed up for: Compromising real-world llm-integrated applications with indirect prompt injection. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security*, pages 79–90. ACM.

LangChain (2024). Langchain documentation. `https://python.langchain.com`.

Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, pages 9459–9474. Curran Associates.

Ollama (2024). Get up and running with large language models locally. `https://ollama.com`.

Park, J. S., O'Brien, J. C., Cai, C. J., et al. (2023). Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22. ACM.

Rao, A. S. and Georgeff, M. P. (1995). Bdi agents: From theory to practice. In *Proceedings of the 1st International Conference on Multi-Agent Systems*, pages 312–319. AAAI Press.

Rosa, R. d. O. (2026). Cerra.ai: Source code repository. `https://github.com/renato0503/A-Multi-Agent-LLM-Architecture-with-Dynamic-Cascading-for-On-Premises`.

Schick, T., Dwivedi-Yu, J., Dessì, R., et al. (2023). Toolformer: Language models can teach themselves to use tools. In *Proceedings of the 37th Conference on Neural Information Processing Systems*. Curran Associates.

Schuster, T., Fisch, A., Gupta, J., et al. (2021). Confident adaptive language modeling. In *Advances in Neural Information Processing Systems*, pages 17456–17472. Curran Associates.

Wei, J., Wang, X., Schuurmans, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Wooldridge, M. (2009). *An Introduction to Multiagent Systems*. John Wiley & Sons, 2nd edition.

Yao, S., Zhao, J., Yu, D., et al. (2023). React: Synergizing reasoning and acting in language models. In *Proceedings of the 11th International Conference on Learning Representations*. OpenReview.