



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

EXECUTIVE SUMMARY OF THE THESIS

Profiling Framework for LLM-Based Data Agents: Quality, Latency, and Energy under Parameter Sensitivity

LAUREA MAGISTRALE IN COMPUTER SCIENCE AND ENGINEERING - INGEGNERIA INFORMATICA

Authors: DAVIDE VALENZISI (245001) OSSAMA EL OUKILI (996171)

Advisor: PROF. DANILO ARDAGNA

Academic year: 2025-2026

1. Introduction

The growing adoption of Large Language Models (LLMs) in enterprise software has opened a concrete path toward natural language interfaces for structured data. A system that translates a business question into a SQL query, executes it, and returns both a textual analysis and a chart makes data-driven decision making accessible to non-technical users. Deploying such systems in production, however, raises practical questions that go beyond accuracy: how long does each inference take, how does it scale with configuration changes, and what is the energy cost of running LLM-backed pipelines on real hardware? These efficiency concerns are central to sustainable AI [1, 2], yet they are rarely measured jointly with quality for LLM-based data agents.

This work addresses both sides of the problem. On the application side, it presents the design and implementation of a *Sales Agent*: an LLM-powered conversational system that lets a business analyst query a synthetic retail dataset in plain English. Given a natural language question, the agent autonomously selects the processing steps, generates and executes a SQL query against Parquet files via DuckDB, produces a textual interpretation of the results, and optionally renders a Matplotlib chart. The agent is orchestrated through LangGraph [3], a framework for stateful graph-based

LLM pipelines, and runs on locally-hosted open-weight models served via Ollama. A hierarchical YAML configuration exposes the inference parameters studied in the experiments — sampling temperature, top- p , top- k , best-of- N candidates, and Chain-of-Thought refinement depth — making each run fully reproducible and configurable without code changes.

On the experimental side, the core contribution is the systematic *profiling* of the agent, which jointly measures output quality, execution time, and per-step energy consumption (tracked with CodeCarbon [4]). A benchmark of 20 analytical questions, stratified across four SQL difficulty levels, evaluates quality along three complementary dimensions: a graded ground-truth table-similarity score for data retrieval, and two LLM-as-judge scores [5] for the textual analysis and the visualization. The evaluation is organised as five step-isolated experiments: the first establishes a quality and energy baseline for three open-weight models (Gemma 4 E4B, Gemma 4 26B, and Mistral Small 3.2 24B), and the following four vary one agent step at a time across sampling hyperparameters, token budgets, and inference-time compute, closing with a Pareto confirmation run. The central finding is that model choice alone does not explain performance: the dominant effects arise from the interaction between model family, agent step, prompt difficulty, and inference-time settings. From this analysis the thesis dis-

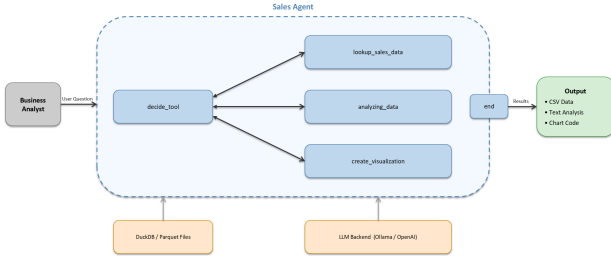


Figure 1: High-level architecture of the Sales Agent.

tills three recommendations for energy-aware LLM data agents — tune hyperparameters per agent step rather than globally, allocate inference-time compute by failure mode, and report accuracy and energy by prompt difficulty.

2. The Sales Agent

The Sales Agent lets a business analyst query a synthetic retail dataset in natural language, without writing SQL. The dataset consists of three Parquet tables — *sales* (daily store-level transactions), *products* (catalog), and *stores* (metadata) — linked by shared keys, so questions can span multiple entities through SQL joins. Given a question, the agent retrieves the relevant data, produces a textual analysis, and optionally renders a chart, returning a CSV table, a natural language interpretation, and executable Matplotlib code.

Architecture. The agent is orchestrated as a stateful directed graph built with LangGraph [3] (Figure 1). A `decide_tool` node acts as a router that, at each iteration, invokes the LLM to choose the next action and loops back until termination. Three processing nodes encapsulate the analytical capabilities: `lookup` generates a SQL query from the prompt and the database schema and executes it via DuckDB; `analysis` interprets the retrieved data in text; and `visualization` produces a chart configuration and the Matplotlib code, which is validated by execution and, on failure, repaired through error-aware refinement when CoT refinement is enabled. A shared `State` object carries the prompt, the data, the generated SQL, the per-step timings and scores between nodes. The LLM backbone is provider-agnostic: open-weight models are served locally via Ollama (used for the agent), while the OpenAI API is reserved for the LLM-as-judge evaluation.

Configuration and inference-time mechanisms. A hierarchical YAML configuration exposes the inference parameters studied in the experiments *per agent step* — sampling temperature, top- p , top- k , token budget, best-of- N candidates, and Chain-of-Thought (CoT) refinement depth — so that each run is fully reproducible without code changes. Two inference-time compute mechanisms are central to the study. *Best-of- N* sampling runs a step N times while sweeping one sampling parameter, then selects the best candidate; during live execution the lookup step uses a ground-truth-free, self-consistency [6] consensus evaluator that picks the candidate with the highest average row-level agreement with the others. *Iterative CoT refinement* feeds a step’s output back into the prompt for up to a configured number of iterations, stopping early on convergence; in error-aware mode it injects the execution error so the model can self-correct.

Benchmark and metrics. The agent is evaluated on a hand-built benchmark of 20 analytical questions, stratified across four SQL difficulty levels (from single-table aggregations to multi-join queries with ratios and year-over-year logic). Quality is measured per step along three complementary dimensions: a graded table-similarity score between the retrieved table and the ground truth for lookup; and two LLM-as-judge [5] scores — factual accuracy and coverage for the analysis, and axis/type/functional correctness for the visualization. Energy is tracked with CodeCarbon [4], which records CPU, GPU, and RAM energy draw both per step and for the whole run, alongside wall-clock latency. This makes quality, latency, and energy directly comparable across configurations.

3. Experimental Design

Three open-weight models are studied, served locally via Ollama: Gemma 4 E4B [7], a compact reasoning model; Gemma 4 26B, its large counterpart; and Mistral Small 3.2 24B [8], an instruction-following model. Each model uses the sampling defaults recommended on its Ollama page (Gemma at temperature 1.0, Mistral at 0.15). Inference is offloaded to the GPU, while LLM-as-judge scoring uses the OpenAI API. Experiments were executed on two GPU servers (NVIDIA L40S and A40) to parallelise data collection: absolute latency and energy are not compared across machines, but each

sweep varies a single factor on a fixed machine, so the rankings and relative trends it captures are unaffected.

The campaign is structured as five *step-isolated* experiments, each isolating one aspect of behaviour:

- **Test 01 — Baseline.** Each model is run with its default configuration on the full 20-prompt benchmark, establishing a quality and energy reference.
- **Test 02 — Sensitivity (OFAT).** Five static sampling hyperparameters (temperature, top- p , top- k , repeat penalty, repeat-window) are swept one-factor-at-a-time, ten values each, separately for every agent step.
- **Test 03 — Token-budget ladder.** The `max_tokens` cap is varied across four levels per step to find the minimum safe budget and any quality sensitivity to it.
- **Test 04 — Compute expansion.** CoT refinement depth is swept (1–4) on the lookup and visualization steps; a separate follow-up (04b) instead studies whether *best-of- N* sampling helps the lookup step, comparing best-of-3 temperature ranges against static temperatures (with a CoT reference).
- **Test 05 — Pareto confirmation.** The most promising *combined* per-step policies from Tests 02–04 are evaluated on the full benchmark to fix the definitive quality-versus-energy frontier.

Latency and energy are treated as two readings of the same cost: on fixed hardware the GPU draws near-constant power during generation, so per-prompt energy is essentially a fixed multiple of wall-clock time (the per-model latency and energy rankings coincide to within about five percentage points). The trade-off analysis therefore uses *energy per prompt* as the single efficiency axis.

4. Results

Baseline (Test 01). At default settings, no model dominates: the three have distinct, *step-specific* failure modes. Mistral Small is the Pareto baseline — the highest strict (end-to-end) quality, 0.827, at the lowest cost (48.9 s and 0.00601 kWh per prompt) — but it collapses deterministically on the two hardest difficulty-4 aggregation prompts. Gemma E4B nearly matches its quality (0.821) with the most balanced per-step profile, but at $1.47\times$ the energy and $1.54\times$ the latency (0.00883 kWh, 75.4 s), and it shows the same

lookup brittleness on hard SQL. Gemma 26B has the strongest data-retrieval and analysis scores yet is held back by a weak visualization step (0.500 against 0.77–0.78 for the others) and the highest cost (0.01667 kWh, 136.0 s), leaving it Pareto-dominated at baseline.

Sensitivity to sampling parameters (Test 02). The one-factor-at-a-time sweep is where the most informative patterns emerge, and its key message is that sensitivity to static sampling parameters is a *model-level* property, not a step-level one. Gemma E4B is by far the most sensitive model: its lookup quality alone spans 0.223 points between the best and worst single setting, more than any step of the other two models. This fragility is also a high-upside, cheap lever — `top_p` = 0.94 raises lookup quality by +0.117 and `csv_iou` by +0.139 for only 5×10^{-4} kWh/prompt — and it is the first concrete sign that the compact model’s weak baseline is a *tuning* problem, not a capacity ceiling. Gemma 26B is also highly tunable but asymmetric: every parameter family improves its weak visualization step (up to +0.068 quality), yet a single bad `top_k` value can collapse lookup quality by -0.123 , and lowering temperature below 1.0 actively hurts its analysis — evidence that the thinking model relies on sampling diversity. Mistral, by contrast, is essentially insensitive: no parameter at any step moves quality by more than +0.007, so its efficiency is robust to misconfiguration but offers almost no tuning upside (its one real risk is a `repeat_penalty` = 1.2 analysis setting that loses 0.099 quality). Two cross-cutting findings stand out: quality-improving settings barely move energy (tuning is mostly a quality concern, not a cost one), and prompt difficulty *amplifies* tuning effects — the largest swings concentrate on difficulty-4 prompts for both thinking models. Figure 2 summarises, per (step, parameter) pair, the best achievable quality delta.

Token budget and inference-time compute (Tests 03–04). The `max_tokens` budget (Test 03) behaves as a per-step *safety bound* rather than a quality dial: it barely affects energy, since the model stops once its output is complete, so raising it is nearly free where a step is genuinely truncated — Gemma 26B visualization gains +0.054 `vis_score` from a larger cap, with no plateau yet visible within the tested range — while overly generous caps can destabilise Gemma E4B into longer, worse output. The decisive lever is *inference-time*

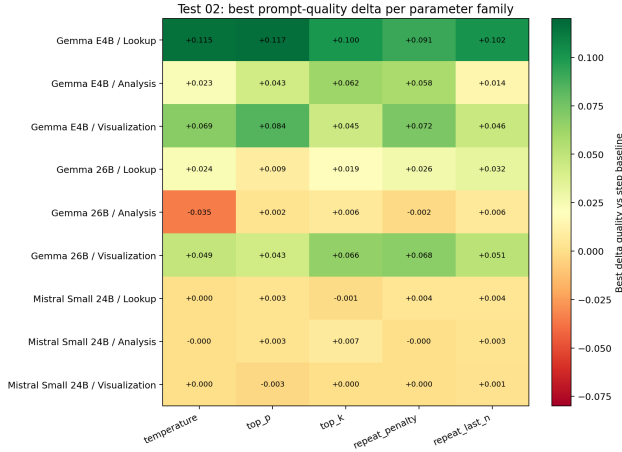


Figure 2: Test 02: best prompt E2E quality delta per (step, parameter) pair for all runs.

compute (Test 04): iterative CoT refinement on the lookup step lifts Gemma E4B by +0.186 quality at depth 2 (*csv_iou* 0.757 $\rightarrow$ 0.954), repairing exactly the SQL brittleness seen at baseline, because the SQL execution result is a feedback signal aligned with correctness; the gain concentrates on the hardest prompts (+0.524 at difficulty 4). On visualization the same mechanism is harmful at every depth, because chart errors are semantic rather than executable, so self-criticism over-edits already-working code. An early-stop mechanism makes the executed depth saturate near 2, so depth 2 is the efficient default. A targeted follow-up (Test 04b) shows that best-of- N temperature sampling can create useful lookup diversity, especially with the medium range [0.6, 1.0], but the no-GT selector still picks the first candidate in most runs. A well-chosen static lookup temperature is therefore the best low-cost option, while CoT remains the stronger quality intervention.

Final trade-off (Test 05). Combining the per-step interventions yields the unified quality–energy frontier of Figure 3. The frontier is *not* a model-size ranking. Mistral with lookup CoT defines the evaluated frontier, from the cheapest useful points around 0.0036 kWh to the highest-quality endpoint: 0.9395 prompt quality, full completion, and 0.00453 kWh per prompt. Gemma E4B — repaired through lookup CoT at a lower temperature with leaner downstream token caps — remains extremely close (0.9371 prompt quality at full completion), but it is slightly more expensive and therefore dominated in the final quality–energy plane. Gemma 26B is explicitly tested with larger visualization-token budgets following Test 03, and

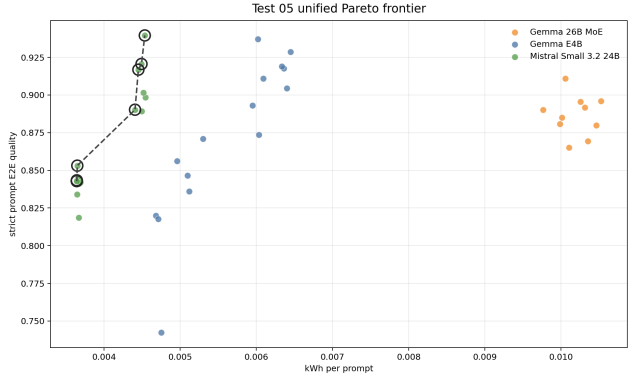


Figure 3: Test 05: unified quality–energy Pareto frontier among the evaluated candidate configurations.

this aligns its weak visualization profile upward (best visualization score 0.674, best prompt quality 0.911), but the repair is incomplete and the energy cost keeps it off the frontier. Splitting by difficulty explains the remaining nuance. Mistral is the best evaluated endpoint on difficulties 1, 2, and 4, while Gemma E4B is the highest-quality endpoint on difficulty 3. The difficulty-4 result does not imply that Mistral would necessarily remain preferable for arbitrarily harder prompts; it shows that, on the evaluated hard-prompt slice, lookup-CoT Mistral is the best observed quality–energy choice, while broader hard-prompt testing remains a natural next step.

5. Conclusions

This work studied the quality–latency–energy trade-off of a multi-step LLM data agent for sales analytics. The central result is that model choice alone does not explain performance: the dominant effects emerge from the interaction between model family, agent step, prompt difficulty, and inference-time parameters. The untuned baseline favours Mistral Small for its efficiency, but the difficulty split reveals what the aggregate hides — it collapses on the hardest SQL aggregations, exactly where targeted lookup repair lets the compact Gemma E4B compete with it. This repair comes specifically from CoT refinement, which exploits executable SQL feedback; best-of- N sampling, by contrast, was limited by the local selector, which still preferred the first candidate in most Best-of-3 runs. After tuning, the quality–energy frontier is defined by Mistral candidates, while Gemma E4B remains a very close high-quality alternative and the larger Gemma 26B improves yet

stays dominated once energy is counted. Notably, the recommended configurations improve substantially over their untuned baselines, so per-step tuning is a genuine improvement rather than a simple quality-for-energy trade.

From this evidence the thesis distills three practical recommendations for energy-aware LLM data agents:

1. **Tune per agent step, not globally.** Sensitivity is a model-level property, and the most rewarding target and safe operating range differ by step, so a single global setting leaves quality on the table.
2. **Allocate inference-time compute by failure mode.** Grant token headroom only to genuinely truncated steps, apply CoT refinement where feedback is verifiable (SQL lookup), and keep generation conservative where errors are semantic (visualization).
3. **Report accuracy and energy by prompt difficulty,** since the best model for simple tasks is not the best for hard analytical questions.

References

- [1] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, 2019.
- [2] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. Green AI. *Communications of the ACM*, 63(12):54–63, 2020.
- [3] LangChain AI. LangGraph: Build stateful, multi-actor applications with LLMs. <https://github.com/langchain-ai/langgraph>, 2024. Accessed: April 2025.
- [4] CodeCarbon Contributors. CodeCarbon: Track carbon emissions from machine learning. <https://github.com/mlco2/codecarbon>. Accessed: January 2026.
- [5] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36. Curran Associates, Inc., 2023.
- [6] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.
- [7] Google DeepMind. Gemma 4 technical report. <https://ai.google.dev/gemma>, 2025. Accessed: May 2025.
- [8] Mistral AI. Mistral small 3.2. <https://mistral.ai/news/mistral-small-3-2>, 2025. Accessed: May 2025.