Relatório Técnico — LLM-SQL Skeleton (v2)

Projeto analisado: `LLM-SQL-skeleton.zip`
Data da análise: 2025-09-04 14:00 (America/Sao_Paulo)
Autor do relatório: ChatGPT (GPT-5 Thinking)

---

1) Visão geral do projeto

Este projeto implementa um **back-end FastAPI** que transforma **perguntas em linguagem natural (PT-BR)** em **consultas SQL seguras** sobre um banco **MySQL** e devolve os resultados em JSON e CSV (via UI web simples).
A estratégia é **template-first** (intents definidas em YAML) com **fallback opcional a LLM via Ollama** apenas para:

- **Classificação de intenção** (quando a heurística não atinge o limiar) e
- **Extração de slots** (datas, N, status, cliente etc.), quando necessário.

Principais componentes:

- **API** (`FastAPI`) com endpoints de saúde, UI de teste e execução de consultas.
- **Registro de intenções** (YAML em `config/intents/*.yaml`) carregado em memória.
- **Motor NLU** (`core/nlu.py`): normalização + heurísticas + *fallback* LLM.
- **Builder SQL** (funções em `api/routes_llm.py`): gera SQL a partir da intenção + slots.
- **Execução segura**: conexão MySQL via `PyMySQL`, *gate* de **EXPLAIN** para bloquear planos caros e regras sintáticas básicas (apenas `SELECT`, sem `UNION`, etc.).
- **UI web** (HTML embutido em `api/main.py` e assets em `static/`) para testar perguntas e baixar CSV.

---

2) Stack, linguagens e versões

Linguagem: Python 3.10 (artefatos `__pycache__/...cpython-310.pyc`).
Framework web: FastAPI 0.111.0 + Uvicorn 0.30.1.
Validação de dados: Pydantic 2.7.1.
Banco de dados: MySQL (driver `PyMySQL 1.1.1`).
Config/parse: PyYAML 6.0.1, `sqlparse 0.5.1`, `python-dotenv >=1.0`.
LLM (opcional): `Ollama >= 0.3.0` com **modelo padrão** `llama3.1:8b` definido em `.env` (`LLM_PROVIDER=ollama`).
Front/UI: HTML+JS minimalista servido pelo próprio FastAPI; `static/mgnet-logo.png`.

Arquivo de dependências: `requirements.txt`.

---

3) Estrutura de diretórios (essencial)

```
api/
  main.py                # bootstrap da API, UI e endpoint /consulta
  routes_llm.py          # /llm/route e /llm/run + builder SQL
config/
  app.yaml               # limites de tempo, CORS, cache, etc.
  datas.yaml             # normalização/formatos de datas PT-BR
  intents/               # intents por domínio (vendas, financeiro,
produção)
    vendas.yaml
    financeiro.yaml
    producao.yaml
  tenants.yaml           # esqueleto p/ multitenancy (não usado no
/consulta atual)
core/
  nlu.py                 # roteamento e extração de slots (heurística
+ LLM opcional)
  llm_provider.py        # integração Ollama (pick intent + extract
slots)
  executor.py, firewall.py, templates.py, router.py, pipeline.py
                         # pipeline modular alternativo
(parcial/experimental)
util/
  intents_loader.py      # carrega/mescla intents *.yaml (com macros)
  dates.py, text.py      # utilitários de datas e normalização de
texto
static/
  mgnet-logo.png
run_local.bat, run_api.bat
README.md
.env                     # credenciais/local de LLM (NÃO commit)
```

Observação: além do fluxo **registrado em `api/routes_llm.py`** (ativo na API atual), existe um **pipeline modular** em `core/` (router + templates + firewall + executor) que pode ser ligado futuramente. No ZIP, ele está presente mas **não é o caminho principal da execução** do endpoint `/consulta`.

---

4) Configuração (arquivos YAML e `.env`)

4.1 `.env` (exemplo encontrado)

```
DB_HOST=…; DB_PORT=3306; DB_USER=…; DB_PASS=****; DB_NAME=…;
DB_CHARSET=utf8mb4
LLM_PROVIDER=ollama
OLLAMA_HOST=http://127.0.0.1:11434
OLLAMA_MODEL=llama3.1:8b
```

- **Recomendação:** nunca versionar senhas; manter `.env` fora do Git. Em produção, usar **variáveis de ambiente/secret manager**.

4.2 `config/app.yaml`

- `limites.timeout_total_ms: 2000` (SLO end-to-end ~2s)
- `limites.timeout_mysql_ms: 1800` (orçamento SQL)
- `pipeline.fallback_llm_orcamento_ms: 250` (abort da LLM se demorar)
- `observabilidade.logs_json: true`
- `seguranca.rate_limit_por_min: 60` e `cors_origens_permitidas`

Nota: alguns itens são **diretrizes** (não há *rate limiter* global implementado no código atual).

4.3 `config/datas.yaml`

- Timezone: **America/Sao_Paulo** / Locale: **pt-BR**
- Formatos aceitos (amostra): `dd/MM/yyyy`, `dd/MM`, `MM/yyyy`, `yyyy-MM-dd`, `yyyy-MM`, `d 'de' MMMM 'de' yyyy`, etc.
- Rótulos como **"hoje", "ontem", "mês atual", "mês anterior", "últimos N dias"** são resolvidos por `util/dates.py`.

4.4 `config/intents/*.yaml`

- **Domínios:** `vendas`, `financeiro`, `producao`.
- Exemplos de intenções (resumo):
  - `vendas.listar_ultimos_N_pedidos` (com slots `data_ini`, `data_fim`, `N`, ordenação, índice em `DT_PVE`).
  - `vendas.topN_clientes_por_valor`, `vendas.contagem_por_periodo`, `vendas.soma_valor_por_periodo`, `vendas.detalhes_pedido`.
  - `financeiro.listar_titulos_pendentes`, `financeiro.pagamentos_por_condicao`, `financeiro.saldo_por_cliente`.
  - `producao.ordens_por_status_no_periodo`, `producao.apontamentos_por_maquina`, `producao.paradas_por_motivo`.
- Cada intenção define: **tabela**, **alias**, **mapeamento de colunas lógicas** (ex.: `numero_pedido: NU_PVE`, `emissao: DT_PVE`), **regras** (ex.: exigir índice, `limit_padrao`, obrigar `LIMIT` em listagens), **ordenação** e **retorno** (`linhas`, `agregado_tabela`, etc.).

4.5 `config/tenants.yaml`

- Suporte planejado a **read-replica** e **IP allowlist** por tenant + caminhos de **catálogo/glossário/regras**.
- **Importante:** o endpoint atual `/consulta` não lê este arquivo; usa a conexão de `.env`. O multitenancy está **esqueleto/planejado**, não ativado.

---

5) API — endpoints e contratos

5.1 Saúde e UI

- `GET /health` — *ping* de saúde.
- `GET /` → redireciona para `GET /app` (UI de teste simples).

5.2 Intent registry (debug)

- `GET /intencoes` — lista as intents carregadas do diretório `config/intents`.

5.3 Execução de consultas (principal)

- `POST /consulta` — recebe pergunta em PT-BR, resolve **intenção + slots**, **gera SQL** e **executa** no MySQL.
  - **Entrada (exemplo):**
  - ```
    {
      "pergunta": "Listar últimos 20 pedidos do mês",
      "use_llm": true
    }
    ```
  - **Saída (exemplo estrutural):**
  - ```
    {
      "intent": "vendas.listar_ultimos_N_pedidos",
      "score": 0.62,
      "slots": {"data_ini":"2025-09-01","data_fim":"2025-09-30","N":20,"ORD_DIR":"DESC"},
      "sql": "SELECT ... FROM mgpve01010 pv WHERE pv.DT_PVE BETWEEN %s AND %s ORDER BY pv.DT_PVE DESC LIMIT %s",
      "executed": true,
      "rowcount": 20,
      "cols": ["NU_PVE","DT_PVE","NU_CLI", "..."],
      "rows": [[1234,"2025-09-03",321,...], ...],
      "rows_dict": [{"NU_PVE":1234,"DT_PVE":"2025-09-03","NU_CLI":321,...}, ...]
    }
    ```
- Endpoints adicionais sob `/llm/…` (definidos em `api/routes_llm.py`):
  - `POST /llm/route` — **só roteia** (classifica intenção e extrai slots) **sem executar SQL**.
  - `POST /llm/run` — **gera SQL** a partir da intenção/slots e **executa** (retorno com `sql`, `cols`, `rows`, `rowcount`).

---

6) Pipeline de processamento (E2E)

1. **Entrada do usuário** (“pergunta”).
2. **Normalização** (`util/text.py`): minúsculas, remoção de acentuação, sinonímia simples (“vendas”→“pedidos”), limpeza de espaços.
3. **Roteamento (NLU)** (`core/nlu.py`):
   - Heurística baseada em **domínio** (`vendas/financeiro/producao`) + **padrões/keywords**; `score` por similaridade (SequenceMatcher).

    - Caso score < *threshold* (por padrão 0.55 no `main.py`), tenta **LLM (Ollama)**:
        - `pick_intent_with_llm`: escolhe 1 chave de intenção dentre rótulos fornecidos.
        - `extract_slots_with_llm`: retorna JSON apenas com slots relevantes.
4. **Resolução de datas/períodos** (`util/dates.py` + `config/datas.yaml`).
5. **Build de SQL** (funções em `api/routes_llm.py`): usa o **mapeamento de colunas** da intenção + **hints de ordenação** (ex.: "primeiros/últimos", "top 10").
6. **Firewall sintático** (básico no builder + validações): somente `SELECT`; **sem `UNION`**, **sem `SELECT *`**, **sem `ORDER BY RAND()`.**
7. ***EXPLAIN gate*** (em `routes_llm.py`): executa `EXPLAIN` e **bloqueia planos** com *full scan* sem índice e cardinalidade muito alta (limite padrão ≈ **500k linhas**).
8. **Execução** (`PyMySQL`): `DictCursor` para obter `rows_dict` e conversão para `cols` + `rows` (lista 2D) para front.
9. **Retorno**: JSON; a UI embutida permite **download em CSV**.

---

7) LLM e IA utilizadas

- **Provedor:** `Ollama` (local). Controlado por `LLM_PROVIDER=ollama` e `OLLAMA_HOST` no `.env`.
- **Modelo padrão:** `llama3.1:8b` (temperatura **0** para previsibilidade).
- **Uso da IA:** *apenas* para **roteamento/slots** quando a regra/heurística não cobrir — **não** há geração de SQL "livre".
- **Orçamento de tempo:** recomendação de abortar fallback LLM acima de **250ms** (definido em `config/app.yaml`).

Alternativas previstas no código (`core/`): um pipeline **templates + firewall + executor** que também poderia acionar LLMs diferentes para **explicar** resultados ou **parafrasear** respostas; atualmente não está ligado no endpoint principal.

---

8) Segurança e governança

- **Somente leitura:** Apenas `SELECT` é permitido; comandos destrutivos são rejeitados.
- **Bloqueios sintáticos:** sem `UNION`, `SELECT *`, `SELECT INTO`, `ORDER BY RAND()` (default).
- ***EXPLAIN gate*:** impede consultas caras (full scan sem índice; ex.: `rows >= 500k` e sem `key`).
- **Limites de resposta:** `max_linhas_resposta: 1000` e `limit_padrao_listagens: 100`.
- **CORS e Rate-limit (diretriz):** configurados em `app.yaml`; implementar *middleware* de rate-limit em produção.
- **Segredos:** `.env` contém credenciais **sensíveis** — **não versionar**; usar secrets/CI.

- **Multitenancy/IP allowlist:** esqueleto em `tenants.yaml` (não ativo no `/consulta`).

---

9) Performance e SLO

- **SLO alvo:** `timeout_total_ms ≈ 2000ms` por requisição.
- **Orçamento SQL:** `timeout_mysql_ms ≈ 1800ms`.
- **Cache de resultados:** `pipeline.cache_ttl_seg: 120` (quando o pipeline modular for ativado).
- **Dica prática:** priorizar **índices** nas colunas usadas em `WHERE/ORDER BY` pelas intents (ex.: `DT_PVE`, `ID_STATUS`).

---

10) Como executar localmente

1. **Instalar dependências**
2. `pip install -r requirements.txt`
3. **Ajustar** `.env` (host/porta/usuário/senha/DB; Ollama opcional).
4. **Subir API**
5. `uvicorn api.main:app --host 0.0.0.0 --port 8080 --reload`

   ou `run_local.bat` / `run_api.bat` (Windows).

6. **Testar** em `http://localhost:8080/app` ou via `http://localhost:8080/docs`.

---

11) Exemplos de uso (cURL)

Roteamento sem executar

```
curl -X POST http://localhost:8080/llm/route -H "Content-Type:
application/json" -d '{
  "utterance": "pedidos desta semana (últimos 20)"
}'
```

Executar consulta completa

```
curl -X POST "http://localhost:8080/consulta?use_llm=true" -H
"Content-Type: application/json" -d '{
  "pergunta": "listar últimos 20 pedidos do mês"
}'
```

---

12) Limitações conhecidas

- **Multitenancy** (`config/tenants.yaml`) **não integrado** ao `/consulta` atual (usa `.env`).
- **Rate-limit** e **auth** não implementados (apenas CORS configurado).
- **Catálogo/Glossário/Regras externas** referenciados em `tenants.yaml` não são lidos no fluxo principal.
- **Validação de tipos/slots** é básica (heurística + LLM opcional); dependerá da **qualidade das intents**.
- **Cobertura de intents**: os YAMLs exemplo são **parciais** (devem ser expandidos para o seu ERP real).

---

13) Recomendações (próximos passos)

1. **Autenticação** (JWT/OAuth2) e **rate-limit** real por IP/usuário.
2. **Ativar multitenancy**: selecionar `tenant` por header/chave API e ler `tenants.yaml` (usa `read_replica` se existir).
3. **Observabilidade**: log estruturado + *request id* + métricas (latência por etapa, erros de EXPLAIN/Firewall).
4. **Catálogo de dados**: gerar `catalogo.json` com tabelas/colunas/índices para melhorar *routing* e validações.
5. **Validações SQL** adicionais: listas de **tabelas/colunas permitidas** por intenção; checagem de **hints de índice**.
6. **Testes** (unitários e integração) com *fixtures* SQL e *golden files*.
7. **UI**: paginação e exportação CSV/Excel nativas; documentação embutida das intents.
8. **LLM**: *guardrails* de tempo e *circuit-breaker*; suporte opcional a **OpenAI**/**Azure OpenAI**, com parametrização de modelo.

---

14) Anexos/Referências internas (do ZIP)

- `README.md` — instruções básicas de execução.
- `config/intents/*.yaml` — exemplos reais de intenções e mapeamentos.
- `core/llm_provider.py` — integração Ollama (modelo padrão: `llama3.1:8b`, temperatura 0).
- `Relatorio_Tecnico_LLM_SQL.pdf` — rascunho anterior existente no ZIP (pode estar desatualizado).

Conclusão

O projeto está **bem estruturado** para consultas SQL seguras via **intents** e oferece **fallback LLM local** (Ollama) para aumentar a robustez do NLU sem abrir mão de **controle** e **desempenho**. Para produção, recomenda-se priorizar **autenticação, rate-limit, multitenancy real, observabilidade** e expandir a **cobertura/qualidade** das intents conforme o seu ERP.