Prompt do Agente Implementador — Roteador + Modelos Especiais (NIM Cloud)

Contexto

Projeto: agente pessoal em **um único chat**, com dinâmica perfeita (rápido, natural, sem travar). Já existe: UI com seleção manual de modelos + RAG completo (vetor/DB vetorial/embeddings/rerank). Falta: **roteador de modelos** e o **mapa de modelos por função especial** (visão, docs, OCR, segurança, fala etc.) usando apenas opções estáveis.

Objetivo

1) Implementar um **Model Router** robusto (principal entrega). 2) Fixar **modelo padrão do chat** em: `meta/llama-3.3-70b-instruct`. 3) **Manter a lista atual** de modelos no UI (não remover nada). Apenas: - adicionar o novo padrão como "Default Recomendado" - adicionar "Auto (Recomendado)" no UI (router ligado) 4) Definir e usar modelos "best-of" por função especial **sem enfraquecer** o chat. 5) Garantir fluxo rápido, resiliente, sem erros: streaming, timeouts, retry com backoff, fallbacks, logs.

A) Mapa de Modelos por Função (Best-Of)

A.1 Chat (texto) — cérebro principal (NÃO usar modelo fraco)

• **default_text_model**

• `meta/llama-3.3-70b-instruct`

• **fallback_text_models (somente se o default falhar)**

• usar **a lista atual do sistema** (preferir 70B equivalentes já existentes)

• regra: **nunca cair automaticamente para <70B** como fallback primário do chat

• **hard_task_models (escalonamento sob gatilhos explícitos)**

• `meta/llama-3.1-405b-instruct` *(se estiver disponível no seu tenant)*
• `nvidia/llama-3.1-nemotron-ultra-253b-v1` *(se estiver disponível no seu tenant)*
• `deepseek-ai/deepseek-r1-0528` *(raciocínio pesado; se estiver disponível)*

Observação: o escalonamento deve ser raro e motivado por gatilhos claros (complexidade alta, falha de qualidade, outputs estruturados extensos, etc.).

A.2 Visão (imagem/anexo visual: prints, fotos, PDF renderizado, slides como imagem)

• **vision_model_id**
• `meta/llama-3.2-90b-vision-instruct` *(se estiver disponível no seu tenant)*

Regra: se o usuário “travou” modelo manualmente e o modelo não suportar visão, roteie **apenas este turno** para `vision_model_id` e depois retorne para o modelo travado (sem mudar o dropdown).

A.3 Multimodal unificado (texto+imagem+áudio) — opcional por feature flag

- **multimodal_omni_model_id**
- `microsoft/phi-4-multimodal-instruct` *(se habilitado/estável no tenant)*

Uso recomendado: quando a mesma mensagem contém **áudio + imagem** no mesmo turno ou quando for explicitamente solicitado um fluxo multimodal. **Não substitui** o chat principal.

A.4 Embeddings (RAG / indexação / busca semântica)

Status: embeddings e pipeline de vetor **já existem** no sistema. Nesta entrega, **não reimplementar** embeddings — apenas **referenciar e integrar** no roteamento/ingestão.

Regra crítica (antes de qualquer embedding): mapear a codebase e eliminar duplicidade 1) Fazer um **inventário único** (source-of-truth) de todos os artefatos indexáveis: repositórios, diretórios, tipos de arquivo, docs, blobs e anexos. 2) Definir chaves canônicas e idempotentes: - `source_id` (repo + path + branch/tag) - `doc_id` (source_id + versão) - `chunk_id` (doc_id + offsets + estratégia de chunking) - `content_hash` (ex.: SHA-256 do conteúdo normalizado) 3) **Dedup obrigatório**: antes de gerar embedding, consultar a base vetorial por `chunk_id` / `content_hash` e **pular** se já existir. 4) Versionamento explícito: - `embedding_model_id`, `embedding_dim`, `chunker_version`, `normalizer_version` - reindex só quando version mudou. 5) Upsert idempotente (nunca inserir duplicado).

Modelos (mantendo o que já temos; sugestões opcionais de melhoria) - **embedding_model_id (texto geral)** - **Manter o atual** (ex.: `nvidia/nv-embed-v1`) — sugestão sólida para embedding geral.

- **embedding_model_code_id (opcional; somente após mapear codebase e deduplicar)**
- `nvidia/nv-embedcode-7b-v1` para corpus de **código** (arquivos .py/.ts/.go etc.) e consultas text↔code.
- **embedding_model_multilingual_id (opcional, se PT-BR e documentos longos forem prioridade)**
- `nvidia/llama-3.2-nemoretriever-300m-embed-v1` (multilíngue, incluindo Português; foco em QA retrieval).
- **embedding_model_multimodal_id (opcional; se vocês indexarem imagens/prints)**
- `nvidia/llama-3.2-nemoretriever-1b-vlm-embed-v1` (embedding multimodal para imagem+texto).

A.5 Rerank (RAG / rerank de candidatos)

- **rerank_model_id**
- `nvidia/rerank-qa-mistral-4b`

A.6 Leitura de documentos (PDF/DOC/PPT; extração com layout)

- **doc_parse_model_id**
- `nvidia/nemotron-parse`

A.7 OCR (quando o doc/anexo for imagem/scan)

- **ocr_model_id**
- `nvidia/ocdrnet` *(se estiver disponível no tenant)*

A.8 Segurança / Moderação (camada antes/depois do LLM)

- **safety_guard_model_id**
- `nvidia/llama-3.1-nemotron-safety-guard-multilingual-8b-v1` *(se habilitado)*

A.9 Fala (ASR/TTS) — serviço

- **ASR (speech-to-text)**
- NVIDIA Riva ASR Multilingual (pt-BR), streaming
- **TTS (text-to-speech)**
- NVIDIA Riva TTS, streaming

Nota: fala é serviço/pipeline; não é o LLM principal.

B) Router (Regras + Política)

B.1 Seleção do usuário (UI)

- UI mantém lista atual.
- Adicionar:
- opção **"Auto (Recomendado)"** (router ON)
- "Default Recomendado" pré-selecionado: `meta/llama-3.3-70b-instruct`
- Se o usuário selecionar manualmente um modelo:
- respeitar SEMPRE para **texto**
- exceção: quando a mensagem exigir visão/áudio e o modelo escolhido não suportar; roteie o turno para modelo especializado e volte ao modelo travado.

B.2 Heurísticas determinísticas (configuráveis)

Sinais: - `has_image_attachment`, `has_audio_attachment`, `has_doc` (pdf/doc/ppt) - `needs_tools` (web/RAG/actions) - `complexity_score` (heurística simples)

Ordem de regras: 1) Se `has_doc`: - rodar `doc_parse_model_id` para extração - se doc for scan/ imagem e parse falhar → usar `ocr_model_id` - indexar/usar RAG (se aplicável) e responder com o modelo de chat (default ou travado) 2) Se `has_image_attachment`: - usar `vision_model_id` - produzir: descrição + entidades + texto extraído (se necessário) - sintetizar/agir no modelo de chat (default ou travado) 3) Se `has_audio_attachment`: - preferir ASR via Riva (pt-BR) → transcript - responder com o modelo de chat (default ou travado) - se também houver imagem no mesmo turno e a feature flag estiver ON, permitir uso do `multimodal_omni_model_id` 4) Se `needs_tools`: - usar o modelo de chat como orquestrador - tool-calling com JSON estrito + validação de schema + auto-repair

5) Se `complexity_score` alto OU falha de qualidade detectada: - escalar para `hard_task_model_id` configurado - registrar motivo no log

B.3 Garantia “um chat só”

- Um único histórico `messages[]`, sem resets.
- Trocas internas de modelo por turno não podem quebrar o contexto.
- Registrar internamente por turno: `model_id`, latência, tokens, fallbacks, tool_calls.

C) Endpoints / Model list / Cache

- Usar endpoint OpenAI-like da NIM Cloud:
- `POST /v1/chat/completions` com **streaming ON**
- Listar modelos:
- `GET /v1/models`
- cache TTL 5–15 min
- se falhar: usar lista local atual e seguir operando

D) Resiliência (Timeout / Retry / Fallback)

- timeout por chamada
- retry com backoff e limites
- fallback:
- se `default_text_model` falhar → tentar `fallback_text_models` (preferência 70B)
- se `vision_model_id` falhar → fallback controlado (ex.: pedir reenvio/ajuste do anexo + resposta curta)

E) Critérios de Aceite (Testes)

1) Auto (Recomendado): texto usa sempre Llama 3.3 70B por padrão. 2) Seleção manual: respeitada sem perder histórico. 3) Anexo imagem: roteia para Vision sem quebrar chat. 4) Docs: parse/OCR + RAG + resposta coerente. 5) Falhas: fallback sem travar UI e sem reset. 6) Streaming: time-to-first-token baixo no chat.

F) Capabilities e Compatibilidade (descoberta automática)

Objetivo: roteamento “sem erro” exige saber, em tempo de execução, o que cada modelo suporta.

Regras: 1) Construir um **Model Capability Registry** via `GET /v1/models` e cache TTL (5–15 min). 2) Para cada modelo, armazenar: `model_id`, limites (contexto), suporte a streaming, suporte multimodal/vision, suporte tool-calling, etc. 3) Antes de enviar a requisição: - validar se o modelo escolhido (manual ou auto) suporta a modalidade (texto/imagem/áudio); - se não suportar, **rotear apenas aquele turno** para o modelo especializado, mantendo a seleção do usuário no UI. 4) Se `GET /`

`v1/models` falhar: - usar o catálogo local atual como fallback; - registrar o evento em log (observabilidade).

G) Excelência de Latência (SLO/SLI + “latency budget”)

Objetivo: sensação de velocidade (dinâmica perfeita) e previsibilidade em produção.

Defina SLO/SLIs (por ambiente: dev/stage/prod): 1) **TTFT/TTFB** (tempo até primeiro token) — alvo e p95/p99. 2) **Tempo total por resposta** — p95/p99. 3) **Erro por endpoint** (4xx/5xx), **taxa de fallback**, **taxa de retry**, **taxa de repair de tool JSON**. 4) **Custo por turno** (tokens in/out) e anomalias.

Regras de execução: - Streaming ON sempre que suportado. - Orquestrador deve iniciar saída cedo; evitar “enrolação”. - Budgets por turno: - limitar chamadas paralelas; - limitar profundidade de tool loops; - cancelar cedo em caso de timeout/sem progresso.

H) Retry, Rate Limit e Circuit Breakers (anti-travamento)

Objetivo: robustez sob 429/5xx sem efeito cascata.

Padrão obrigatório: 1) Retry apenas para erros transitórios: `408`, `429`, `5xx` (e falhas de conexão). 2) **Truncated exponential backoff + jitter**. 3) **Retry budget** por request (limite máximo de tentativas e tempo total). 4) Respeitar idempotência: tool actions com efeito colateral exigem confirmação do usuário; evitar re-execução automática. 5) Rate limiter cliente: - impedir thundering herd; - limitar concorrência por usuário e por sessão. 6) Circuit breaker por dependência (vector DB, rerank, parse, web fetch, etc.): - abrir circuito em falhas repetidas; - fallback seguro; - retomar com half-open.

I) Contrato de Tool Calling (schema-first, validação e reparo)

Objetivo: tool-use confiável, sem “quebrar” a dinâmica do chat.

Regras: 1) Toda tool call deve ter **schema JSON explícito** (request/response). 2) **Validação** (schema) antes de executar qualquer ação. 3) Se inválido: - 1 tentativa de **auto-repair** (pedir ao modelo para corrigir JSON para o schema); - se ainda inválido: não executar; responder com alternativa segura. 4) Desabilitar tool calls paralelas quando a consistência for crítica. 5) Sanitização/escapes no retorno de ferramentas (nunca executar output “como comando”).

J) Segurança (OWASP LLM Top 10 — controles obrigatórios)

Objetivo: agente com anexos/web/ferramentas sem vulnerabilidades clássicas.

Políticas mínimas: 1) **Prompt Injection**: qualquer conteúdo externo (web, docs, OCR, RAG) é **DADO não confiável**, nunca instrução. 2) **Insecure Output Handling**: outputs do LLM não podem ser executados diretamente (shell/sql/http) sem validação e escapes. 3) **Excessive Agency**: limitar autonomia: - classes de ações: auto-safe, confirm-required, blocked; - confirmação explícita para ações com efeito colateral.

4) **DoS**: limites de tamanho, limites de loops, limites de tokens e anexos. 5) **Sensitive Info**: redaction e políticas de não vazamento (logs, prompts, outputs). 6) **Supply chain**: allowlist de ferramentas, endpoints e domínios; bloquear SSRF.

K) Gestão de Contexto e Memória (sem “inchar” e sem perder naturalidade)

Objetivo: chat estável e fluido ao longo do tempo.

Regras: 1) Separar “lanes” de memória: - **Short-term**: últimos turnos relevantes. - **Long-term**: perfil/ preferências do usuário (resumo compacto). - **Task state**: estado do plano/tarefa atual. 2) Token budget por mensagem e por sessão; compressão automática: - sumarização incremental quando exceder budget; - armazenar “memória longa” como fatos curtos. 3) RAG controlado: - nunca injetar contexto enorme; - sempre limitar a K trechos + rerank.

L) Governança do Catálogo de Modelos (mudanças sem quebrar produção)

Objetivo: você tem muitos modelos; a governança evita regressões.

Regras: 1) Allowlist por função: - chat_default, hard_task, vision, parse, ocr, safety, etc. 2) “Pinning” por ambiente (dev/stage/prod) e rollout: - canary (ex.: 5%) + métricas; - rollback automático por SLI. 3) Toda troca de modelo/heurística exige passagem na suíte de avaliação (ver seção M).

M) Qualidade e Avaliação (sem regressão)

Objetivo: manter o chat “Gemini-like” e útil ao longo do tempo.

Regras: 1) Criar um **conjunto de cenários pt-BR** (golden set): - humor leve / conversa casual; - planejamento e execução; - anexos (doc + imagem); - tool calls; - falhas simuladas (timeouts, 429, parse falhando). 2) Rodar regressão antes de mudanças em router/modelos. 3) Logging de “bad cases” com redaction.

N) Política de Estilo (pt-BR natural, adaptável, sem comprometer segurança)

Objetivo: conversa natural (brinca/zoa) com controle.

Regras: 1) Tom padrão: - pt-BR natural; - humor contextual moderado; - direto quando o usuário pedir tarefa séria. 2) O “tom” nunca pode influenciar decisões de segurança, ferramentas ou confirmações. 3) Quando houver dúvida, fazer 1 pergunta objetiva; caso contrário, agir com assertividade.

O) Health Checks, Readiness e Degradação Elegante

Objetivo: não travar quando dependências caírem.

Regras: 1) Implementar health endpoint do seu backend e checar dependências: - NIM endpoint (ready/models), vector DB, rerank, parse/ocr, web fetch. 2) Em falhas: - circuit breaker; - fallback seguro; - mensagens curtas e objetivas ao usuário (sem detalhes internos).

Entregáveis

1) Model Router + config. 2) Integração mínima com pipeline atual. 3) README interno (como ajustar regras, trocar modelos, testar). 4) Logs/telemetria por turno.