DOCUMENTAÇÃO DO PROOF OF CONCEPT (POC)

1 PASSO 1: VALIDAÇÃO DA CONECTIVIDADE E NÓ BÁSICO (PRÉ-LANGGRAPH)

Este passo inicial tinha como objetivo validar a conectividade com a API do Gemini, garantir a segurança das credenciais e refinar a Engenharia de Prompt para que a LLM gerasse a estrutura de saída rígida esperada para a orquestração futura.

1.1 CONECTIVIDADE E ROBUSTEZ DO CLIENTE

Objetivo: Estabelecer a comunicação funcional entre o ambiente Python e o modelo Gemini e testar a robustez contra falhas de rede.

1.1.1. Conclusões da Conectividade:

1. **Conectividade:** A comunicação com o modelo `gemini-2.5-flash` foi estabelecida com sucesso, usando a biblioteca oficial do Google GenAI.
2. **Segurança:** A utilização do arquivo `.env` para armazenar a `GEMINI_API_KEY` foi validada como a prática correta.
3. **Robustez (Tratamento de Erros):** A necessidade de implementar a lógica de *Exponential Backoff* foi confirmada em logs de erro. Esta função demonstrou ser crucial para a resiliência do sistema, permitindo que a aplicação se recuperasse automaticamente de limites de taxa temporários.

1.2 VALIDAÇÃO DA ESTRUTURA E DA ENGENHARIA DE PROMPT

Objetivo: Garantir que a saída da LLM obedeça à estrutura rígida (três pontos fixos) essencial para a leitura por máquina e a tomada de decisão pelo LangGraph.

1.2.1. Análise da Estrutura de Saída (Gatekeeper): O prompt foi projetado para impor uma persona e, o mais importante, uma **Estrutura de Saída Rígida**, que atua como o *Gatekeeper* do LangGraph.

Tabela 1: Estrutura de Saída Rígida e Relação com o LangGraph

Estrutura de Saída Exigida	Por que é Crucial para o LangGraph
1. Avaliação:	O LangGraph usará este texto como **Gatekeeper** (Porteiro) para decidir o próximo nó.
2. Justificativa:	Fornece feedback qualitativo ao aluno.
3. Sugestão de Correção:	Ajuda o aluno a corrigir o código.

1.2.2. Resultados da Correção Simples (POC Inicial): Os testes iniciais validaram que a LLM conseguia seguir a estrutura rígida e fornecer um feedback relevante.

Tabela 2: Resultados do POC - Validação Inicial do Prompt

Caso de Teste	Avaliação da LLM (Passo 1)	Conclusão
Código Correto	Avaliado como "Certo".	**VALOR AGREGADO:** Capacidade de ir além do enunciado, sugerindo boas práticas.
Código com Erro	Avaliado como "Parcialmente Certo"(na primeira tentativa).	**VALIDAÇÃO ESTRUTURAL:** O formato rígido foi respeitado, validando o Prompt.

2 PASSO 2: VALIDAÇÃO DA ORQUESTRAÇÃO COM LANGGRAPH

Este passo teve como objetivo testar a execução sequencial do nó de correção dentro de um grafo do LangGraph, garantindo que o sistema pudesse processar diferentes entradas de código de forma robusta e diferenciada.

2.1 RESULTADOS DA EXECUÇÃO SEQUENCIAL DOS CASOS DE TESTE

A execução de três casos de teste (Correto, Errado e Parcialmente Certo) em sequência foi concluída com sucesso, demonstrando a capacidade do LangGraph de gerenciar o estado e invocar a LLM repetidamente.

Tabela 3: Resultados da Execução Sequencial no LangGraph (Passo 2)

Caso de Teste (Código)	Avaliação da LLM	Conclusão da Análise
1. Código Correto	Certo	**CONFIRMAÇÃO:** A LLM manteve a capacidade de mentor, sugerindo boas práticas (construtor, validação de negativos).
2. Código com Erro	**Errado**	**VALIDAÇÃO CRÍTICA:** Confirma a detecção de falhas graves de POO (**Encapsulamento** e **Lógica** incorreta) e a obediência à Estrutura Rígida.
3. Código Parcialmente Certo	Parcialmente Certo	**ROBUSTEZ:** O teste foi concluído com sucesso, apesar de ter esbarrado no limite de taxa da API (recuperação via Exponential Backoff), e a avaliação identificou corretamente a falta de validação de valores de entrada.

3 PASSO 3: VALIDAÇÃO DA LEITURA E ESTUDO DE INPUT

Este passo tinha como objetivo principal validar a entrada de dados do mundo real por meio da leitura de arquivos e avaliar a melhor forma de enviar esse código para o modelo.

3.1 IMPLEMENTAÇÃO DA LEITURA DE ARQUIVOS

1. A leitura do enunciado do exercício (arquivo `.txt`) e do código do aluno (arquivo `.java`) foi implementada com sucesso no script `poc-leitura-arquivos-langgraph.py` utilizando a função nativa de I/O do Python.

2. O script LangGraph foi refatorado para receber o conteúdo lido dos arquivos como entrada no estado inicial, preparando o sistema para testes com o *dataset* completo.

3.2 AVALIAÇÃO DA ESTRATÉGIA DE ENVIO DE CÓDIGO ("ANEXO")

Foi avaliada a melhor estratégia para "anexar"o arquivo de código Java à LLM.

3.2.1. Conclusão da Estratégia: A melhor prática para o envio de código-fonte foi confirmada como a leitura do conteúdo do arquivo para uma string (‘str‘) e a inclusão dessa string dentro do prompt da LLM.

- **Eficiência:** Evita a sobrecarga de codificação (Base64) necessária para anexos binários (multimídia), que é desnecessária para código-fonte.

- **Confiança da LLM:** Enviar o código como texto formatado (usando *code fences*

Markdown, ‘‘‘`java` e ‘‘‘) é o método mais confiável para garantir que o modelo interprete o conteúdo como código Java.

- **Simplicidade do Grafo:** Mantém a estrutura de estado (‘CorrectionState‘) simples, utilizando apenas strings para tráfego de dados.

4 PASSO 4: CORREÇÃO DE PROJETO COM MÚLTIPLOS ARQUIVOS

O objetivo desta etapa era testar a capacidade da LLM de corrigir um projeto com múltiplos arquivos, exigindo correlação e análise de consistência entre classes.

4.1 ANÁLISE DO PROJETO MULTI-ARQUIVO

Projeto Testado: Sistema de Calculadora Simples.

4.1.1. Resultados e Validação: A estratégia de concatenação dos arquivos (`ClassePrincipal.java` e `ClasseAuxiliar.java`) em uma única string, com delimitadores claros, demonstrou-se altamente eficaz:

- **Consistência Arquitetônica:** A LLM identificou o erro mais grave: a incompatibilidade entre a `ClassePrincipal` (que chamava o método `somar` de forma estática) e a `ClasseAuxiliar` (onde o método `somar` era de instância).
- **Erro Lógico:** O erro lógico (`return x - y;` em vez de `return x + y;`) dentro de `ClasseAuxiliar` foi corretamente detectado.
- **Feedback Integrado:** A sugestão de correção forneceu duas opções completas, corrigindo o código em **ambos os arquivos** de forma integrada e coerente, comprovando que o modelo tratou o input como um projeto único.
- **Conclusão do Passo 4:** O fluxo POC está validado para corrigir projetos Java multi-arquivo, superando a dificuldade de contexto de código distribuído.

5 PASSO 5: AVALIAÇÃO DE VIABILIDADE ECONÔMICA

O objetivo deste passo foi realizar uma análise de custo-benefício e viabilidade do projeto, com foco nas ferramentas de Inteligência Artificial Generativa.

5.1 CUSTOS DA API DO GEMINI

O modelo de preços da API do Gemini é baseado no pagamento por uso (*pay-as-you-go*), processado através do sistema *Cloud Billing*. A cobrança é realizada com base em vários fatores:

Contagem de Tokens de Entrada (prompt e contexto), Contagem de Tokens de Saída (resposta gerada), e Tokens em Cache.

O `gemini-2.5-flash` oferece o melhor custo-benefício com o preço de entrada mais baixo, sendo ideal para tarefas de correção de alto volume, enquanto o `gemini-2.5-pro` oferece maior capacidade de raciocínio para projetos mais complexos a um custo mais elevado.

Solicitações de API que falham com erro 400 ou 500 não são cobradas. O uso do Google AI Studio para testes e desenvolvimento **permanece sem custo financeiro**, mesmo com o faturamento ativado. Créditos do Google Cloud são aceitos para o pagamento do uso da API Gemini.

5.2 LICENÇA ACADÊMICA E LIMITAÇÕES

Os limites de taxa são as métricas que regulam o número de solicitações que podem ser feitas à API Gemini em um determinado período. Esses limites são medidos em três dimensões principais e aplicados por projeto (não por chave de API):

- **Solicitações por Minuto (RPM):** O número máximo de requisições que podem ser feitas em 60 segundos.
- **Tokens por Minuto (TPM):** O volume máximo de tokens de entrada (prompt) que pode ser processado em 60 segundos.
- **Solicitações por Dia (RPD):** O número máximo de requisições que podem ser feitas em 24 horas (redefinidas à meia-noite do Horário do Pacífico). Esta é a principal restrição do Nível Grátis.

Exceder qualquer um desses limites (ex: 21 solicitações em um minuto com limite de RPM de 20) aciona um erro de limitação de taxa. Esses limites variam drasticamente dependendo do *status* do projeto (Grátis, Nível 1, Nível 2 ou Nível 3), sendo mais restritos para modelos experimentais e de prévia.

Dinâmica dos Limites de Taxa. Os limites de taxa da API Gemini dependem do nível de cota do projeto e são atualizados automaticamente à medida que o nível e o status da conta mudam. É crucial notar que os limites de taxa especificados não são garantidos, e a capacidade real pode variar dependendo da demanda do sistema e da região.

1. **Licença Acadêmica:** Não há uma "licença acadêmica"gratuita formalmente divulgada. O uso para pesquisa se enquadra no *Free Tier* ou no plano pago (*pay-as-you-go*). O plano pago é fundamental, pois garante que os dados do usuário e prompts **não serão utilizados para treinar os modelos do Google**.

2. **Limitações de Cota:** O *Free Tier* (**20 RPD**) mostrou-se impraticável para a POC (conforme logs do Passo 2). A solução para a escalabilidade é a ativação do faturamento no projeto do Google Cloud, removendo o limite de Requisições por Dia (RPD) e elevando as cotas de Requisições por Minuto (RPM) e Tokens por Minuto (TPM) para níveis que permitem uso intensivo.

5.3 ALTERNATIVAS DE LLMS

O mercado oferece diversas alternativas robustas para a correção de código, com variação notável em custo e capacidade de raciocínio. A tabela a seguir compara o custo por 1 milhão de tokens (1M T) dos principais concorrentes de baixo custo e alta performance, sendo a taxa de saída (Output) a métrica mais relevante para o custo final do feedback.

Tabela 4: Comparação de Custos por 1 Milhão de Tokens (1M T) - Modelos de Baixo Custo

Modelo	Custo Entrada (US$/1M T)	Custo Saída (US$/1M T)	Vantagem Competitiva
DeepSeek-V3.2	$0.28	**$0.42**	Mais baixo custo de saída (feedback), ideal para correção em massa e otimização de custo.
OpenAI GPT-5 mini	$0.250	$2.00	Mais baixo custo de entrada e alta velocidade (concorrente direto do Flash).
Google Gemini 2.5 Flash	$0.30	$2.50	Ótimo custo-benefício e janela de contexto de 1M (modelo usado na POC).

- **OpenAI (Modelos de Alta Performance):** GPT-5.2 Pro (Entrada $21.00/1M T, Saída $168.00/1M T) e Gemini 2.5 Pro (Entrada $1.25/1M T, Saída $10.00/1M T) são as opções de raciocínio mais avançado, sendo mais adequadas para problemas arquitetônicos complexos, mas com custo significativamente mais alto.
- **LLaMA/Mistral:** Implementação local (*self-hosted*) é a alternativa para custo zero de API, mas exige alto investimento inicial em infraestrutura de hardware (GPUs).