Relatório

📊 Relatório de Validação Técnica: Killua Bot

Projeto: Automação de Diagramas UML via Telegram com Google Gemini e n8n
Motor de Renderização: PlantUML (Padrão) / Mermaid.js (Experimental)
Data: Novembro 2025
Responsável: João Pedro

---

1. Resumo Executivo

O **Killua Bot** foi desenvolvido para resolver o gargalo de documentação em projetos de software, utilizando Inteligência Artificial Generativa para transformar código fonte e requisitos em diagramas UML visuais instantâneos.

Este relatório documenta a eficácia da ferramenta através de um teste comparativo rigoroso entre a modelagem humana (manual) e a modelagem automatizada (IA), validando a capacidade do bot de atuar como um auxiliar de arquitetura confiável.

---

2. Metodologia de Teste (O Desafio "Humano vs. Máquina")

Para validar a precisão da ferramenta, foi adotada a estratégia de **"Gold Standard Manual"**:

1. **O Gabarito (Ground Truth):** O responsável (João Pedro) desenhou manualmente os diagramas de Classes, Atividades e Casos de Uso baseados no script `codigo-teste.py`. Esses desenhos serviram como a "verdade absoluta" da lógica do sistema.
2. **A Automação:** O mesmo código foi submetido ao Killua Bot em dois cenários distintos de prompt.
3. **A Comparação:** O resultado da IA foi cruzado com o gabarito manual para avaliar precisão semântica, identificação de erros e qualidade visual.

O Objeto de Teste (`codigo-teste.py`):

- Sistema de vendas com Herança (POO).
- Loops de validação de input (`while`).
- Tratamento de exceções (`try/except`).

---

3. Análise Comparativa de Resultados

3.1. Diagrama de Classes (`/genClass`)

Critério	Modelagem Manual (João Pedro)	Killua Bot (PlantUML + Prompt Refinado)
Estrutura	Correta, focada nas relações macro.	**Superior.** Além das relações, identificou estereótipos técnicos (ex: `<<utility>>` para Main) e tipagem de dados (`str` , `float`).
Herança	Mapeada corretamente.	**Perfeita.** Usou a notação UML estrita (seta triangular vazia) para `Product` e suas filhas.

> **Veredito:** O bot foi capaz de gerar um diagrama com rigor técnico igual ou superior ao manual, adicionando metadados de tipagem que muitas vezes são esquecidos no desenho à mão.

3.2. Diagrama de Atividades (`/genFlow`)

Este foi o ponto crítico de comparação, pois o código continha loops complexos de validação.

- **O Desenho Manual:**
  Focou na lógica principal do menu, simplificando alguns tratamentos de erro para manter a legibilidade (foco no "Caminho Feliz").
- **O Resultado do Killua Bot:**
  Surpreendeu pela fidelidade ao código. O bot criou uma **Raia (Partition)** específica para a função `get_paint_details` , isolando visualmente o loop de validação de cor da tinta. Ele mapeou os caminhos de exceção (erro de input) com precisão, mostrando fluxos de retorno que o desenho manual abstraiu.

> **Conclusão:** O diagrama da IA serviu como uma "auditoria" do código, revelando complexidades ciclomaticas que passaram despercebidas na modelagem manual inicial.

3.3. Diagrama de Casos de Uso (`/genCase`)

- **O Desenho Manual:**
  Correto, destacando a relação de `<<include>>` na compra de tinta.
- **O Resultado do Killua Bot:**
  Através do uso de Engenharia de Prompt Contextual, o bot replicou exatamente a lógica

do manual. Identificou a fronteira do sistema e posicionou a seta de dependência (`include`) na direção correta (Base -> Incluído), um erro comum em IAs não treinadas.

---

4. Análise das Engines: PlantUML vs. Mermaid

Durante os testes de estresse, foi identificado um comportamento heterogêneo no motor **Mermaid.js**, variando conforme o tipo de diagrama solicitado:

4.1. O Caso de Sucesso: Diagramas de Classes (Estrutural)

- **Performance:** ⭐ ⭐ ⭐ ⭐ ⭐ (Excelente).
- **Observação:** O Mermaid processou diagramas de classes com perfeição, independentemente da complexidade do prompt. A estrutura declarativa deste diagrama (`class A` , `relationship B`) mostrou-se compatível com a geração da IA.

4.2. O Gargalo: Diagramas Comportamentais (Atividades e Casos de Uso)

- **Performance:** ⭐ ⭐ (Instável).
- **O Problema:** Para diagramas que exigem lógica de fluxo, loops ou agrupamentos (`subgraphs`), o Mermaid apresentou alta taxa de falha, retornando frequentemente o erro `Syntax error in text` .
- **Causa Raiz:** A sintaxe do Mermaid para fluxos é "Strict" (rígida). Se a IA gera um caractere de seta ligeiramente errado (ex: `-->` em vez de `-.->`) ou esquece um ponto e vírgula em um loop aninhado, a renderização quebra totalmente.

4.3. A Solução Robusta: PlantUML

Em contraste, o motor **PlantUML** (baseado em Java) demonstrou resiliência superior. Ele foi capaz de ignorar pequenos erros de sintaxe da IA e renderizar a lógica correta dos diagramas comportamentais onde o Mermaid falhou.

Decisão Arquitetural: O Killua Bot mantém o **PlantUML como motor padrão** para garantir a confiabilidade em fluxos complexos, mas oferece o Mermaid como opção secundária, recomendada especificamente para Diagramas de Classes ou fluxos simples.

---

5. Conclusão Final

O experimento comprovou que o **Killua Bot**, quando guiado por prompts contextuais (Chain-of-Thought), atinge um nível de precisão comparável à de um desenvolvedor júnior/pleno na modelagem de diagramas.

A comparação com os diagramas feitos à mão por João Pedro evidenciou que a IA não apenas "desenha", mas é capaz de interpretar regras de negócio implícitas no código. O projeto foi validado com sucesso e está apto para auxiliar no fluxo de trabalho de documentação de software.

Relatório gerado automaticamente a partir da validação técnica do projeto Killua.