UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
PRÓ-REITORIA DE GRADUAÇÃO
DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIA
CURSO INTERDISCIPLINAR EM TECNOLOGIA DA INFORMAÇÃO

Francisco Afonso Matos Pereira Júnior

Um estudo sobre o uso de LLMs na Arquitetura de Software

PAU DOS FERROS
2025

Francisco Afonso Matos Pereira Júnior

Um estudo sobre o uso de LLMs na Arquitetura de Software

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Tecnologia da Informação.

Orientador: João Batista de Souza Neto, Prof. Dr.

PAU DOS FERROS

2025

P436e Pereira Júnior, Francisco Afonso Matos.
Um estudo sobre o uso de LLMs na arquitetura de software / Francisco Afonso Matos Pereira Júnior. – 2025.
68 f. : il.

Orientador: João Batista de Souza Neto.
Monografia (graduação) – Universidade Federal Rural do Semi-árido, Curso de Tecnologia da Informação, 2025.

1. LLMs. 2. Arquitetura de software. 3. Engenharia de prompt. 4. Diretrizes. I. Souza Neto, João Batista de, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automáto em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Pau dos Ferros
Bibliotecária: Erlanda Maria Lopes da Silva
CRB: 15/966

Francisco Afonso Matos Pereira Júnior

Um estudo sobre o uso de LLMs na Arquitetura de Software

Monografia apresentada à Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Tecnologia da Informação.

Defendida em: 05 / 08 / 2025.

BANCA EXAMINADORA

João Batista de Souza Neto, Prof. Dr. (UFERSA)
Presidente

Reudismam Rolim de Sousa, Prof. Dr. (UFERSA)
Membro Examinador

Huliane Medeiros da Silva, Profa. Dra.(UFERSA)
Membro Examinador

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus por ter me conduzido sempre nos melhores caminhos e me acompanhado em cada tribulação enfrentada. Sem ele, nada seria possível.

Agradeço a meu pai, Francisco Afonso Matos Pereira e minha mãe, Francisca Sandra Pereira, que sempre me incentivam e nunca medem esforços para que eu busque crescer cada vez mais. Além disso, eles sempre me ajudaram a levantar em cada tropeço durante essa caminhada, devo tudo a eles.

Agradeço a minha família, irmão, avós, tios e tias, primos e primas, que sempre contribuíram para o meu crescimento pessoal e profissional, e sempre buscaram ajudar de alguma forma.

Agradeço ao meu Orientador Prof. Dr. João Batista de Souza Neto, primeiramente, por ter me dado a oportunidade de ser seu orientado, por compartilhar seus valiosos conhecimentos e por sua disponibilidade para resolver as dificuldades durante o nosso trabalho.

Agradeço a Banca Examinadora por dedicarem seu tempo e conhecimentos para a avaliação deste trabalho.

Agradeço a todos os meus amigos por todo o apoio e momentos compartilhados, que contribuíram grandemente para minha jornada.

Agradeço a minha amiga, namorada e companheira, Letícia Mirely Santiago Santos, por ter me dado luz em todos os momentos, me incentivando sempre quando eu queria apenas desistir e celebrando pequenas conquistas, mostrando sempre o valor de cada passo.

RESUMO

A arquitetura de software é uma atividade crucial no desenvolvimento de sistemas, definindo a estrutura e as qualidades que impactam diretamente o sucesso do projeto. Nesse contexto, os Modelos de Linguagem de Grande Escala (LLMs) emergem com um potencial significativo para auxiliar e otimizar diversas tarefas de arquitetura. Este trabalho investigou o potencial e as limitações de LLMs no apoio às atividades de arquitetura de software. O objetivo geral foi analisar o uso atual de LLMs nesse domínio, identificando potenciais usos, limitações e estratégias para formular decisões arquiteturais qualificadas. A metodologia inclui uma revisão de literatura sobre fundamentos de arquitetura de software, LLMs na engenharia de software e engenharia de *prompt*, resultando na proposição de um guia de diretrizes para o uso de LLMs em tarefas da Arquitetura de Software. Experimentos práticos foram conduzidos, simulando tarefas como proposta e avaliação de arquiteturas, comparação de soluções e geração de documentações da arquitetura, com foco na aplicação de estratégias de engenharia de *prompt*. Os resultados revelaram a capacidade dos LLMs em compreender requisitos complexos e gerar propostas estruturadas, bem como artefatos padronizados. Contudo, também evidenciaram limitações como generalização em detalhes específicos, interpretações restritas de contexto e erros de sintaxe, reforçando a necessidade de supervisão humana e *prompts* refinados. O estudo conclui que a colaboração entre humano e LLM, em que a IA atua como co-agente inteligente e o arquiteto humano mantém o julgamento crítico, é a abordagem mais promissora para alavancar o poder dessas tecnologias, apesar das limitações inerentes.

Palavras-chave: modelos de linguagem de grande escala (LLMs); arquitetura de software; engenharia de *prompt*; diretrizes;

ABSTRACT

Software architecture is a crucial activity in systems development, defining the structure and qualities that directly impact the success of the project. In this context, Large Language Models (LLMs) emerge with significant potential to assist and optimize various architectural tasks. This work investigated the potential and limitations of LLMs in supporting software architecture activities. The overall objective was to analyze the current use of LLMs in this domain, identifying potential uses, limitations, and strategies for formulating qualified architectural decisions. The methodology includes a literature review on software architecture fundamentals, LLMs in software engineering, and prompt engineering, resulting in the proposal of a guideline for the use of LLMs in Software Architecture tasks. Practical experiments were conducted, simulating tasks such as architecture proposal and evaluation, solution comparison, and architecture documentation generation, with a focus on applying prompt engineering strategies. The results revealed the ability of LLMs to understand complex requirements and generate structured proposals, as well as standardized artifacts. However, they also highlighted limitations such as generalization in specific details, restricted interpretations of context, and syntax errors, reinforcing the need for human supervision and refined prompts. The study concludes that collaboration between humans and LLMs, in which AI acts as an intelligent co-agent and the human architect maintains critical judgment, is the most promising approach to leverage the power of these technologies, despite their inherent limitations.

Keywords: large language models(LLMs); software architecture; prompt engineering; guidelines.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
ADR	Registros de Decisões de Arquitetura
AKM	Gerenciamento do Conhecimento Arquitetônico
API	Interface de Programação de Aplicações
ATAM	Método de Análise de Trade-offs de Arquitetura
DB	Banco de Dados
DLQ	Fila de Mensagens Mortas
IA	Inteligência Artificial
LLM	Modelo de Linguagem de Grande Escala
MVC	Modelo-Visão-Controlador
ORM	Mapeador de Objeto Relacional
REST	Transferência de Estado Representacional
SAAM	Método de Análise de Arquitetura de Software
STI	Sistema Tutor Inteligente
SWOT	Forças (Strengths), Fraquezas (Weaknesses), Oportunidades (Opportunities) e Ameaças (Threats)
UML	Linguagem de Modelagem Unificada

SUMÁRIO

1 INTRODUÇÃO

A Arquitetura é um dos pilares centrais na construção de sistemas computacionais complexos, pois define a estrutura organizacional de componentes do sistema, suas interações, restrições e diretrizes técnicas. Segundo Sommerville (2019), a concepção arquitetural é uma etapa inicial e estratégica no desenvolvimento de software, funcionando como um elo entre os requisitos do sistema e seu projeto técnico. É nesse momento que se definem os elementos estruturais centrais e suas relações, o que influencia significativamente a qualidade geral do sistema e sua capacidade de evolução.

Ao longo dos anos, o processo de definição arquitetural tem sido feito por profissionais altamente experientes, baseando-se em padrões já consolidados, boas práticas e raciocínio contextual especializado. Ademais, a complexidade dos sistemas mais novos e a necessidade de respostas ágeis para a construção de projeto de Arquitetura do Software motivam a busca de novas abordagens que auxiliem os arquitetos. Nesse contexto, surgiram os Modelos de Linguagem de Grande Escala (LLMs), por exemplo, o ChatGPT, como possível solução. De acordo com o artigo de Ozkaya (2023), um LLM trata-se de um modelo de rede neural treinado a partir de grandes volumes de dados, como livros, artigos, sites e códigos. Através desse treinamento, o LLM consegue correlacionar as informações necessárias para entregar ao usuário uma resposta coerente, em linguagem natural. Conforme destaca Ozkaya (2023), esses modelos, treinados com grandes volumes de dados, oferecem uma nova interface de interação entre desenvolvedores e sistemas complexos, abrindo possibilidades para automatizar atividades que antes exigiam conhecimento técnico especializado. Essa transformação, embora repleta de oportunidades, também traz desafios e riscos que precisam ser analisados rigorosamente.

1.1 Problematização

Segundo Ganesh e Sahlqvist (2024), apesar do avanço dos LLMs em tarefas automatizadas de desenvolvimento de software, a aplicação dessas tecnologias na fase de projeto da Arquitetura de Software ainda é embrionária. Atualmente, não existem muitos estudos aprofundados que investigam as potencialidades e limitações desses modelos na formulação de decisões arquiteturais, especialmente, em contextos reais e complexos. Apesar

do seu potencial, a aplicação de LLMs na Arquitetura de Software ainda é repleta de desafios. Segundo Eisenreich *et al.* (2024), os LLMs atuais apresentaram resultados promissores na geração de conceitos de domínio, mas exigiram o uso de técnicas, como a engenharia de *prompts*, e supervisão humana para obter um resultado coerente. Outrossim, a variabilidade das respostas, a dificuldade de rastreabilidade dos *Architectural Decision Records* (ADRs) e a falta de critérios próprios para validar as soluções sugeridas também são alguns dos problemas.

Além disso, os estudos sistemáticos que avaliam o desempenho dessas ferramentas em contextos reais de projetos arquitetônicos são escassos. A maioria das pesquisas explora casos simulados ou conduz experimentos com pequenas amostras, como estudantes em ambientes controlados, o que limita a generalização dos resultados. De acordo com a pesquisa de Jahic e Sami (2024), alguns participantes indicaram que os LLMs mostram baixa confiabilidade por diversos motivos, dois deles afirmaram que apesar de usarem o mesmo conjunto de perguntas para o LLM, nesse caso, o ChatGPT, obtiveram respostas distintas. Além disso, um dos participantes relatou que não houve possibilidade de obter resposta em algumas ocasiões, evidenciando limitações na consistência das respostas fornecidas pela ferramenta. Assim, torna-se evidente a necessidade de compreender melhor quais estratégias utilizar para melhorar a consistência das decisões arquitetônicas geradas por LLMs. Por isso, faz-se necessário o desenvolvimento de pesquisas científicas que explorem e avaliem o uso de LLMs no apoio à Arquitetura de Software.

Essa necessidade de pesquisa é ainda mais pertinente quando se considera a natureza complexa e de alto risco da Arquitetura de Software. A concepção de uma arquitetura é uma etapa repleta de desafios e decisões de impacto duradouro. Segundo Sommerville (2019), um dos principais problemas é o elevado custo associado a mudanças estruturais tardias, pois a refatoração da arquitetura pode exigir a modificação da maior parte dos componentes do sistema. O autor ainda aponta que, por essa razão, as decisões iniciais são particularmente críticas.

Sommerville (2019) também descreve o projeto arquitetural como uma atividade inerentemente criativa e não padronizada, que depende mais da experiência do arquiteto do que de um processo formalizado. Esse cenário eleva a responsabilidade do profissional, que frequentemente precisa arbitrar conflitos entre requisitos não funcionais, como o impasse entre desempenho e manutenibilidade, buscando um equilíbrio onde cada decisão representa um compromisso. Um desafio final, apontado pelo autor, é a dificuldade de validar o projeto de forma antecipada, pois seu verdadeiro teste só ocorre quando o sistema está em uso,

Sommerville (2019). Essa incerteza eleva o risco do projeto, pois o fracasso de uma decisão pode só ser notado quando o custo para corrigi-la já se tornou proibitivo.

1.2 Justificativa

A Arquitetura de Software é reconhecida por seu papel determinante na qualidade, escalabilidade e manutenção de sistemas modernos, sendo responsável por estruturar soluções técnicas com base em padrões, *frameworks* e decisões estratégicas. A adoção de arquiteturas adequadas e de múltiplos padrões de projeto pode, conforme apontado por Ganesh e Sahlqvist (2024) busca proporcionar maior eficiência no tratamento de diferentes casos de uso e melhorar significativamente a qualidade do produto de software.

Apesar dos avanços recentes, os desafios associados ao uso de LLMs na formulação de decisões arquiteturais abrem espaço para investigações que busquem estratégias mais eficazes de aplicação. Questões como a variabilidade das respostas, a falta de mecanismos formais de validação e a possibilidade de omissões estruturais evidenciam a necessidade de estudos que promovam maior confiabilidade e robustez no uso dessas ferramentas em cenários reais (Dhar; Vaidhyanathan; Varma, 2024).

Diante disso, este projeto se justifica pela necessidade de compreender e avaliar criticamente o uso de LLMs na Arquitetura de Software, com foco em como estratégias de engenharia de *prompt* podem influenciar a qualidade, a rastreabilidade e a aplicabilidade das decisões geradas. Ao investigar essa interseção, a pesquisa visa contribuir tanto para a literatura acadêmica quanto para práticas profissionais, oferecendo subsídios que favoreçam a criação de sistemas mais escaláveis, eficientes e alinhados a padrões técnicos consolidados.

1.3 Objetivos

Esta seção está organizada da seguinte forma: a Subseção 1.3.1 define o objetivo geral deste trabalho e a Subseção 1.3.2 apresenta os objetivos específicos.

1.3.1 Objetivo geral

Analisar o uso de Modelos de Linguagem de Larga Escala (LLMs) nas atividades relacionadas à Arquitetura de Software de modo a identificar os potenciais usos e limitações, além de estratégias para apoiar e formular decisões arquiteturais qualificadas.

1.3.2 Objetivos específicos

- Investigar o estado da arte sobre o uso de LLMs em tarefas relacionadas à Arquitetura de Software;
- Identificar os principais de desafios e limitações observados na literatura e nos estudos empíricos quanto à aplicação de LLMs nesse domínio;
- Propor um guia de melhores práticas para o uso de LLMs em tarefas da Arquitetura de Software;
- Avaliar experimentalmente o uso de LLMs em atividades da Arquitetura de Software.

1.4 Organização do trabalho

A organização deste trabalho, além da parte introdutória, segue a seguinte estrutura: no Capítulo 2, é apresentada o referencial teórico desse campo de estudo; no Capítulo 3, são apresentados os trabalhos relacionados a pesquisa; no Capítulo 4, é apresentada a metodologia que será utilizada no desenvolvimento do estudo; o Capítulo 5 apresenta as diretrizes para o uso de LLMs na Arquitetura de Software; o Capítulo 6 traz os resultados de um estudo de caso e uma análise crítica dos dados obtidos na pesquisa. Por último, o Capítulo 7 apresenta as considerações finais e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo, serão apresentados os principais conceitos indispensáveis para compreensão deste trabalho.

2.1 Arquitetura de software

De acordo com Valente (2020), a Arquitetura de Software pode ser compreendida sob duas perspectivas principais. Primeiramente, ela se refere ao projeto de alto nível de um sistema, concentrando-se na organização e nas interfaces de unidades maiores, como pacotes, componentes ou subsistemas, em vez de classes individuais, sendo importante que esses componentes sejam relevantes para os objetivos centrais do sistema. Em segundo lugar, tratando de uma forma mais geral, a Arquitetura de Software engloba as decisões mais relevantes do sistema, pois uma vez tomadas, são difíceis de serem revertidas, incluindo escolhas como linguagem de programação e bancos de dados, além da definição de módulos principais.

Segundo Sommerville (2019), essa atividade é uma das etapas iniciais mais importantes do processo de *design*, pois estabelece a base sobre a qual o sistema será construído. A arquitetura permite transformar requisitos abstratos em uma estrutura técnica concreta, o que facilita a comunicação entre os envolvidos no projeto e contribui para a tomada de decisões técnicas ao longo do desenvolvimento. Ainda conforme o autor, a arquitetura influencia significativamente nas características não funcionais do sistema, como desempenho, manutenibilidade, escalabilidade, robustez e modularidade, sendo especialmente crítica em sistemas de grande escala, onde decisões estruturais têm impacto direto na viabilidade e na qualidade do software entregue.

Nesse cenário, a Arquitetura de Software deixa de ser apenas uma etapa do desenvolvimento e passa a atuar dinamicamente e de forma interativa e estratégica. Sua aplicação correta é fundamental para alinhar os objetivos de negócio, ao mesmo tempo se adaptando às necessidades contextuais e os avanços tecnológicos.

Para aprofundar a compreensão sobre esse tema, as próximas subseções abordam três aspectos centrais da Arquitetura de Software: Visões arquiteturais (2.1.1), Padrões arquiteturais (2.1.2), e a documentação da Arquitetura (2.1.3).

2.1.1 Visões arquiteturais

De acordo com Sommerville (2019), os modelos de Arquitetura de Software são fundamentais tanto para o entendimento e a definição de requisitos quanto para a documentação

e desenvolvimento de sistemas. Porém, devido à sua complexidade, apenas uma representação é insuficiente para abordar todos os aspectos da arquitetura. Por isso, é comum utilizar diferentes visões arquiteturais, cada uma focando em aspectos específicos, como a estrutura modular, a execução dos processos ou a distribuição física dos componentes.

Uma das abordagens mais reconhecidas citado por Sommerville (2019) para esse propósito é o modelo 4+1 de Kruchten (1995), que organiza a Arquitetura de Software em quatro visões principais, interligadas por meio de casos de uso e cenários. A visão lógica descreve as abstrações fundamentais do sistema, como classes e objetos, relacionando os requisitos com as entidades funcionais. A visão de processo mostra em tempo de execução e suas interações, sendo útil para avaliar características como desempenho e concorrência. A visão de desenvolvimento aborda a organização do código e a divisão de responsabilidades entre equipes, e por fim a visão física que apresenta a distribuição dos componentes de software na infraestrutura de hardware, combinadas essas visões permitem uma compreensão mais ampla do sistema e facilita a comunicação entre os envolvidos no projeto.

2.1.2 Padrões arquiteturais

Padrões arquiteturais são soluções reutilizáveis para problemas recorrentes no projeto de sistemas de software. Eles fornecem estruturas organizacionais comuns que podem ser aplicadas em diferentes contextos, permitindo a padronização de decisões arquiteturais, a redução da complexidade e a melhoria da comunicação entre os membros da equipe de desenvolvimento. Segundo Sommerville (2019), os padrões de arquitetura não são implementações prontas, mas são modelos bem sucedidos em sistemas anteriores que descrevem como os componentes devem ser organizados e como devem se comunicar. Os padrões ajudam arquitetos a tomar decisões fundamentadas sobre a estrutura geral do projeto e facilitam a reutilização de soluções demonstradas eficazes.

Dentre os principais padrões arquiteturais apresentados por Sommerville (2019), destacam-se:

- Arquitetura em camadas: organiza o sistema como um conjunto de camadas, cada uma fornecendo serviços a camada superior e consumindo serviços da camada inferior. Esse estilo é visto principalmente em sistemas de informação e aplicações web, onde há separação entre interface com o usuário, lógica de aplicação e acesso a dados. Sua principal característica é o isolamento de responsabilidades, facilitando a manutenção e evolução do sistema.
- Modelo-Visão-controlador (MVC): separa a interface (visão), a intermediação entre visão e modelo (controlador) e a lógica de negócio (modelo). De acordo com Sommerville

(2019), esse amplamente utilizado em sistemas interativos, especialmente aplicações com interface gráfica, pois permite atualizar a interface sem alterar a lógica de negócios.

- Arquitetura Cliente-Servidor: Segue o princípio da separação entre clientes que solicitam serviços e servidores que os fornecem. Esse estilo é predominante em sistemas distribuídos, como aplicações web, onde clientes interagem com servidores por meio de requisições de rede. Sommerville (2019) destaca simplicidade de implementação e a divisão clara de responsabilidade como principais pontos positivos.
- Arquitetura de Duto e Filtro (*Pipes and Filters*): estrutura o sistema como uma cadeia de componentes (filtros), conectados por canais de comunicação(dutos), pelos quais os dados seguem de forma sequencial. Cada filtro realiza uma transformação sobre os dados. Esse padrão é útil principalmente em sistemas que usam processamento contínuo ou transformação de dados em etapas, como compiladores ou sistemas de fluxo de dados.

A escolha de um padrão depende do contexto do sistema, dos requisitos do projeto e das prioridades de atributos de qualidade, sejam eles manutenibilidade, desempenho, escalabilidade, entre outros. É muito comum encontrar a combinação de padrões em um único projeto para que as necessidades do sistema sejam atendidas da melhor forma.

2.1.3 Documentação da arquitetura

Durante o processo de desenvolvimento de software, decisões arquiteturais são tomadas com frequência e podem impactar significativamente a estrutura, o comportamento e os atributos de qualidade do sistema. Entretanto, muitas vezes essas decisões não são documentadas de forma adequada, o que dificulta a comunicação entre as partes envolvidas e compromete a rastreabilidade entre requisitos, arquitetura e implementação. Segundo Harrison, Avgeriou e Zdun (2007), registrar somente a decisão final é insuficiente. É necessário documentar também o problema enfrentado, as alternativas consideradas, a motivação por trás da escolha realizada e as possíveis consequências.

A prática usada para lidar com essa falta de informação é o uso de *Architectural Decision Records* (ADRs), Esses registros representam uma forma sistemática e leve de documentar decisões importantes, facilitando o compartilhamento do conhecimento arquitetônico. Os autores destacam que a ausência dessa documentação causa a vaporização do conhecimento arquitetônico, ou seja, resultando em perda de informações críticas que ficam apenas como conhecimento tático armazenado na mente dos arquitetos.

Além do registro textual, a documentação de Arquitetura pode ser reforçada através de modelos visuais, facilitando a compreensão entre os envolvidos no projeto. Entre as abordagens mais utilizadas, destacam-se o modelo C4 e *Unified Modeling Language* (UML).

O modelo C4, proposto por Simon Brown (2022), é uma abordagem hierárquica para modelar a Arquitetura de Software em quatro níveis de abstração: Contexto, *Containers*, Componentes e Código. Essa estrutura permite que diferentes públicos, stakeholders técnicos e não técnicos compreendam a arquitetura de acordo com seu nível de interesse e necessidade de detalhe. O modelo C4 é especialmente útil em ambientes ágeis, pois favorece uma documentação leve, atualizável e centrada na comunicação clara da estrutura do sistema.

Já a UML é uma linguagem de modelagem mais consolidada, desenvolvida inicialmente para representar sistemas orientados a objetos. Ela oferece diversos tipos de diagramas, como diagrama de classes, sequência, casos de uso, componentes, entre outros. No entanto, Sommerville (2019) observa que apesar da eficácia da UML para a descrição do sistema em estágios mais detalhados, ela não foi concebida para descrever abstrações arquiteturais de nível mais alto. O autor argumenta que elementos como classe são muito próximos da implementação, questionando o seu uso na fase inicial, preferindo o uso de notações informais nesse estágio, por serem mais adequadas a discussões entre os stakeholders. Ainda assim, a UML é muito valiosa quando se deseja documentar formalmente a arquitetura, ou adotar o desenvolvimento dirigido por modelos.

2.2 Modelos de linguagem de grande escala (LLMS)

LLMs são modelos criados a partir de algoritmos de aprendizado de máquina que são treinados a partir de grandes volumes de dados textuais, como livros, sites e fóruns, por exemplo, para aprender a reconhecer padrões e relações contextuais em linguagem natural. Assim, esses modelos tornam-se capazes de compreender o significado de frases em diferentes contextos e realizar tarefas como geração de texto, tradução automática, classificação e respostas a perguntas sem exigência de adaptações específicas (Caelen; Blete, 2023).

Esses modelos permitem que os computadores processem, interpretem e gerem linguagem humana, possibilitando uma melhor comunicação entre o homem e a máquina. A partir de um texto de entrada (que é chamado de *prompt*), o LLM utiliza esse conhecimento previamente adquirido para prever as palavras seguintes mais prováveis e, desta forma, possam gerar respostas significativas ao texto de entrada. Atualmente, dentre os principais LLMs, destacam-se o GPT-4[1] desenvolvido pela OpenAI, o Gemini[2], projetado pela Google DeepMind e o DeepSeek, modelo de código aberto projetado pela Hangzhou DeepSeek AI.[3]

Congruente a isso, por compartilharem uma base de funcionamento parecida, ao receberem um *prompt*, esses modelos utilizam o conhecimento aprendido para prever as palavras

[1] https://chatgpt.com/
[2] https://gemini.google.com/
[3] https://chat.deepseek.com/

mais prováveis a seguir, gerando respostas coesas e contextualmente relevantes. No entanto sua eficácia depende fortemente da qualidade das instruções fornecidas, ou seja, os *prompts* devem ser claros, bem direcionados e por vezes construídos de forma iterativa para alcançar os melhores resultados.

2.3 Engenharia de *prompt*

De acordo com Caelen e Blete (2023), a engenharia de *prompt* é uma disciplina emergente que foca em desenvolver as melhores práticas para a elaboração de entradas (*prompts*) eficientes capazes de orientar o comportamento dos LLMs. Um *prompt*, é a mensagem de entrada escrita em linguagem natural que serve como instrução para que o modelo gere uma resposta. Essa entrada pode ser uma pergunta, um comando, uma descrição de contexto ou exemplos, e sua estruturação é influência direta na qualidade e precisão da resposta gerada.

Ademais, os autores destacam que a engenharia de *prompt* envolve o domínio de boas práticas para criar instruções claras, bem direcionadas e contextualizadas, para obtenção de respostas programaticamente úteis. Assim, a atividade se diferencia de simples comandos por exigir refinamento, testes iterativos e adaptação à sensibilidade dos modelos a variações mínimas de linguagem.

White *et al*. (2023a) descrevem a engenharia de *prompt* como um processo programável que permite controlar o comportamento dos LLMs por meio de instruções precisas em linguagem natural. Dentre as principais técnicas da engenharia de *prompts* apresentadas por White *et al*. (2023a), podemos citar:

- Controle de formato: Utiliza o *Meta Language Creation Pattern*, explicando sua semântica para definir linguagens personalizadas de forma específica para futuras interações e o *Output Automater Pattern* para gerar scripts automatizados, que moldam o formato de saída que o LLM vai apresentar.
- Padronização didática: Emprega o *Template Pattern* através de uma especificação de modelo de saída, que o LLM vai preencher com o conteúdo para resultar em saídas consistentes e claras, contribuindo para a organização e didática do conteúdo gerado.
- Gestão comunicacional: Aplica o *Persona Pattern* atribuindo um personagem ou profissional específico para o LLM para modular o tom e estilo de resposta do LLM.
- Fundamentação de respostas no material fornecido: Utiliza-se do *Fact Check List Pattern*, pois exige que o LLM gere os fatos nos quais a saída apresentada foi baseada. O *Reflection Pattern* contribui também para a fundamentação de respostas mesmo que de forma indireta, pois exige que o LLM auto avalie sua saída e identifique erros.

- Fluxo de interação: Exemplificado pelo *Flipped Interaction Pattern,* que inverte o fluxo de interações tradicionais, fazendo com que o LLM faça perguntas ao usuário para coletar informações adicionais e entregar uma resposta mais robusta.

Além disso, White *et al.* (2023b) aprofunda o conceito de padrões de *prompt* definidos como soluções reutilizáveis para problemas recorrentes na interação com LLMs, fazendo analogia aos padrões de software. Mesmo que o artigo não aborde os padrões em categorias, é possível identificá-las de acordo com o seu propósito. Entre elas, destacam-se os padrões voltados a melhoria de qualidade do código, promovendo a modularização e separação de responsabilidades, também é possível detectar padrões voltados a redução de acoplamento, orientando o modelo a abstrair dependências externas, nota-se também o grupo voltado a organização de interação, ajudando a reconhecer como organizar vários *prompts* em sequências lógicas para obter respostas mais precisas. No contexto da arquitetura, os destaques ficam para o *Architectural Possibilities Pattern*, que propõe alternativas de *design* com base nos requisitos fornecidos, o *Change Request Simulation Pattern*, que analisa o impacto de modificações na arquitetura proposta. Esses padrões demonstram como os LLMs podem ser usados estrategicamente se apoiados ao raciocínio técnico e à tomada de decisão.

3 TRABALHOS RELACIONADOS

A crescente integração de Modelos de Linguagem de Grande Escala (LLMs) em aplicações de software tem motivado diversas investigações sobre suas implicações arquiteturais. No contexto da Arquitetura de Software, esses modelos estão sendo explorados como agentes colaborativos no apoio à tomada de decisão, geração de soluções de arquiteturas e na automação de atividades. Os trabalhos analisados neste capítulo compartilham como foco o estudo do uso de LLMs em tarefas arquiteturais, destacando o papel da engenharia de *prompt* como elemento central para orientar a interação efetiva entre humanos e os modelos. Por meio de abordagens que vão desde a geração assistida de decisões até o mapeamento de padrões de integração, essas pesquisas evidenciam os caminhos para uma prática de arquitetura mais responsiva, interativa e eficiente impulsionadas por um uso proveitoso das IAs.

3.1 Can LLMs generate architectural design decisions? - an exploratory empirical study

O artigo de Dhar, Vaidhyanathan e Varma (2024), investigou o potencial dos LLMs na geração automatizada de *Architectural Design Decisions* (ADDs), que são escolhas cruciais que moldam a estrutura e o comportamento de um sistema de software a partir de descrições contextuais, com foco na criação de *Architectural Decision Records* (ADRs).

Os autores partem da constatação de que, apesar da importância das ADRs para a gestão do conhecimento arquitetural (AKM), sua adoção na prática ainda é limitada. Isso acontece prinicpalmente a respeito do esforço necessário para documentar as decisões, a ausência de ferramentas adequadas de suporte e a falta de clareza sobre o que exatamente deve ser registrado. O estudo propõe que os LLMs com a capacidade de interpretar o contexto e gerar textos estruturados, podem ser capazes de automatizar parcialmente a documentação de decisões arquitetônicas.

A metodologia adotada consistiu em um estudo empírico exploratório, no qual diferentes abordagens de geração de texto com LLMs foram testadas: *zero-shot prompt*, em que o modelo gera decisões com base exclusivamente no contexto fornecido, *few-shot prompt*, no qual são apresentados ao modelo alguns exemplos de pares contexto-decisão, *fine-tuning*, em que o modelo é ajustado com um conjunto de dados específico de ADRs. O material do estudo foi construído a partir de 95 ADRs coletados através de uma pesquisa Web com palavras-chave *Architecture Decision Record*, nesse caso a maioria dos repositórios foram encontrados no GitHub, os resultados foram avaliados por métricas de similaridade textual padrões, como:

ROUGE-1, BLEU, METEOR e BERTScore (F1), que apresenta um nível considerável de equivalência ao julgamento humano.

Os resultados apresentam que:

- A abordagem *zero-shot* permite a geração de ADRs consideráveis, principalmente se envolver modelos de grande porte como o GPT-4, mas ainda exige revisão e auxílio humano para assegurar consistência e completude.
- O *few-shot prompting* melhora o desempenho de alguns modelos menores e mais econômicos, tornando possível alcançar resultados comparáveis ao GPT-4, mas no geral o impacto ainda é inconsistente.
- O *fine-tuning* foi a abordagem com os melhores resultados, permitindo que modelos compactos como o Flan-T5-base apresentassem desempenho comparável a modelos bem mais robustos como (GPT 3.5 e GPT-4), com a vantagem de poderem ser executados localmente, ganhando privacidade e viabilidade operacional.

A principal conclusão do estudo é que, embora os LLMs ainda não substituem completamente o julgamento humano na geração de decisões arquiteturais, eles podem funcionar como ferramentas de apoio e podem acelerar o processo. Modelos menores e customizados por *fine-tuning* aparecem como opções interessantes para organizações que buscam equilíbrio entre desempenho, custo e controle de dados. Este trabalho reforça, assim, a viabilidade da integração de LLMs nas fases de documentação e apoio à decisão dentro do processo de Arquitetura de Software, alinhando-se a outras iniciativas que investigam o papel desses modelos em atividades cognitivas, analíticas e estruturais da engenharia de software.

3.2 Towards human-bot collaborative software architecting with ChatGPT

O artigo de Ahmad et al. (2023) explora o uso do ChatGPT como um agente colaborativo, no contexto da Engenharia de Software, conhecido como *DevBot.* O estudo surge devido a ausência de processos padronizados, a dificuldade de unificar a visão de projeto com os *stakeholders* e a falta de arquitetos experientes, propondo que sistemas baseados em LLMs funcionem como parceiros no processo de Arquitetura de Software.

A proposta de colaboração entre humano e robô é centrada na ideia de que o ChatGPT atue como co-arquiteto, auxiliando em tarefas de análise, síntese e avaliação arquitetural. A pesquisa se apoia em um estudo de caso com o aplicativo CampusBike, no qual o ChatGPT auxilia na formulação de requisitos de qualidade, na síntese de soluções por meio da geração de scripts UML e aplicação de padrões de *design*, e na avaliação da arquitetura por meio do método SAAM (*Software Architecture Analysis Method*).

Além dos aspectos técnicos o estudo aborda outras questões críticas, como:

- **Variação de respostas:** o comportamento aleatório do ChatGPT pode comprometer a reprodução de decisões concisas. A solução proposta é o uso de diálogo interativo e a supervisão humana para garantir consistência.
- **Ética e propriedade intelectual:** a geração automática de especificações e código levanta as questões sobre direitos autorais e conformidade legal.
- **Tendências e confiança:** os LLMs podem apresentar distorções decorrentes de seus dados de treinamento, o que afeta a qualidade de recomendações de arquitetura.

Porém, o estudo reconhece limitações importantes, como a generalização restrita decorrente do uso de um único estudo de caso de complexidade moderada. Entretanto, o estudo sugere caminhos futuros promissores, como a integração do ChatGPT em ambientes de desenvolvimento mais complexos e a realização de experimentos controlados com equipes reais e arquitetos.

Em resumo, o trabalho de Ahmad et al. destaca que o ChatGPT não apenas pode auxiliar, mas também enriquecer os processos existentes na Arquitetura de Software, desde que utilizado de forma crítica, ética e supervisionada, reforçando ainda mais o papel dos LLMs como co agentes inteligentes no *design* arquitetural, e não apenas como ferramentas passivas.

3.3 From requirements to architecture: An AI-based journey to semi-automatically generate software architectures

O artigo apresentado por Eisenreich et al.(2024), propõe um método novo para gerar candidatos a Arquitetura de Software a partir de requisitos textuais, utilizando LLMs em um processo semi automático iterativo, que combina geração automática com intervenção e refinamento humano, com motivação no desafio de projetar arquiteturas que atendam aos requisitos de qualidade, principalmente quando trata-se de domínios complexos, onde arquitetos modelam apenas uma arquitetura devido ao tempo e conhecimento limitado.

Como foi mencionado anteriormente, a motivação do trabalho partiu do reconhecimento de que projetar Arquiteturas de Software com alta qualidade é um processo complexo, especialmente em domínios onde os requisitos são incompletos, contraditórios ou evoluem rapidamente. Arquitetos menos experientes, tendem a buscar conhecimentos e tecnologias familiares, limitando a qualidade de soluções possíveis. O método que o artigo propõe visa acabar com esse problema seguindo seis etapas, são elas:

- Geração automática de modelos de domínio e cenários de caso de uso a partir dos requisitos.
- Refinamento manual dos artefatos por arquitetos reais.
- Derivação automática de candidatos a arquitetura, incluindo ADRs.

- Avaliação automática baseada em métricas e abordagens como Método de Análise de *Trade-offs* de Arquitetura (ATAM).
- Refinamento manual dos candidatos.
- Seleção da arquitetura final com base em análise comparativa.

O processo é dividido assim para ser interativo, permitindo revisões contínuas à medida que os requisitos mudam, evitando custos desnecessários. Através de uma análise exploratória o com LLaMA2 70-B e GPT-3.5, os autores identificaram a capacidade dos LLMs em identificar conceitos relevantes, porém, foram encontradas dificuldades, a primeira foi que ao contrário de modelar o domínio como previsto inicialmente os LLMs tentaram modelar o próprio sistema, também notou-se uma dificuldade dos LLMs em criar um código PlantUML razoável. Após isso foi feita uma segunda interação onde a utilização de engenharia de *prompt* foi essencial para a melhorar a qualidade dos resultados.

Em síntese, os autores concluem que uma solução totalmente automatizada ainda é inviável atualmente, porém uma abordagem semi automática apresenta um grande potencial, mostrando-se muito eficiente como suporte a Arquitetura de Software. A proposta de equilibrar IA com supervisão humana, permite ganhos em produtividade sem sacrificar a qualidade técnica de soluções.

3.4 State of practice: LLMs in software engineering and software architecture

O artigo de Jahic e Sami (2024), investiga o uso atual e futuro de LLMs em empresas de engenharia de software, bem como a capacidade desses modelos em auxiliar tarefas complexas como o *design* arquitetônico. O estudo busca preencher a lacuna de pesquisa empírica sobre a adoção de LLMs no setor e seus benefícios para arquitetos de software.

O ponto de partida do estudo é a afirmação de que, apesar de os LLMs estarem revolucionando a geração de código, depuração e resumo de documentação, o uso no *design* arquitetural ainda precisa de comprovações robustas. A arquitetura como disciplina emergente para os LLM e altamente dependente de conhecimento tático, enfrenta problemas na documentação, representação de soluções e automatização. Nesse sentido, o estudo gira em torno de responder 3 questionamentos:

- Quais são os aspectos positivos e negativos do *design* arquitetural assistido por LLM?
- Como as empresas de software utilizam os LLMs atualmente e quais são os benefícios e desafios enfrentados?
- Como as empresas planejam utilizar os LLMs no futuro e quais são os benefícios e desafios esperados?

A metodologia combinou experimentos com uma pesquisa. Para o primeiro questionamento, foram realizados 50 experimentos com o GPT 3.5 em cinco projetos de software focando em geração de diagramas C4(Contexto, Contêiner e Componente) e sugestões de ferramentas. Para os outros 2 questionamentos, foi conduzida uma pesquisa com 15 empresas europeias, coletando dados sobre a adoção atual e futura dos LLMs, e seus benefícios e desafios percebidos.

Respondendo o primeiro questionamento, O ChatGPT demonstrou potencial para gerar soluções de *design* arquitetônico, mas com limitações. A taxa de sucesso na geração de resultados visuais foi menor que a textual. As soluções geradas, embora por vezes inspiradoras e capazes de introduzir novos padrões de *design*, apresentaram problemas como falta de reprodutibilidade, inconsistência lógica, mistura de níveis de abstração, ausência de detalhes sobre interações e partes interessadas, e problemas de qualidade visual. Os participantes da pesquisa classificaram o ChatGPT como uma ferramenta "inspiradora", mas não como uma técnica de engenharia confiável devido à sua imprevisibilidade.

Para responder o segundo questionamento, em relação à adoção atual nas empresas, 90% das organizações relataram já utilizar o ChatGPT em seu cotidiano, principalmente para atividades como assistência à codificação, geração de código, resumo de documentação e aumento de produtividade. Outras ferramentas como CodePilot e Grammarly GO também aparecem como opções complementares. Entre os principais benefícios identificados estão a agilidade, a redução de esforço e a disponibilidade de sugestões variadas de solução. Porém desafios como baixa qualidade de alguns resultados gerados por LLMs, riscos de vazamento de propriedade intelectual, direitos autorais sobre códigos e artefatos gerados e alto esforço de verificação e validação humana ainda estão presentes.

Para responder ao terceiro questionamento, as empresas esperam utilizar LLMs para geração de casos de teste, aumento da produtividade, geração de documentação e melhoria da qualidade das soluções são os benefícios mais esperados, mas mantém as preocupações com os riscos já identificados.

O estudo conclui que, embora os LLMs como o ChatGPT demonstrem potencial para auxiliar na Engenharia de Software e Arquitetura, especialmente na geração de *design*, eles ainda não se qualificam como ferramentas de engenharia totalmente confiáveis devido à sua imprevisibilidade e à necessidade de validação humana extensiva. No entanto, existe uma grande expectativa quanto ao futuro dos LLMs na área devido ao enorme potencial apresentado.

3.5 Discussões

Essa seção estabelece a correlação entre os principais resultados dos trabalhos de pesquisa analisados e as diretrizes estabelecidas na proposta de diretrizes para o uso de LLMs nas tarefas de Arquitetura de Software, justificando as escolhas e tarefas presentes no projeto. A integração de LLMs em aplicações de software tem motivado diversas investigações sobre suas implicações arquiteturais. A engenharia de *prompt* surge como um componente central nesse processo, atuando como elo entre a intenção humana e a geração automática de artefatos técnicos. No entanto, embora os LLMs ofereçam capacidades impressionantes, a literatura indica que sua utilização exige cautela e mediação baseadas em critérios técnicos.

Na Tabela 1 podemos ver um sumário dos trabalhos relacionados apresentados neste capítulo:

Tabela 1 - Resumo dos trabalhos relacionados para apresentação

Referência	Foco	Principal Contribuição	Limitação Identificada
Dhar, Vaidhyanathan e Varma (2024)	Geração automatizada de Registros de Decisão de Arquitetura (ADRs).	LLMs podem acelerar a documentação de decisões, mas ainda não substituem o julgamento de um arquiteto humano.	A geração de ADRs consistentes e completas ainda exige revisão e auxílio de um profissional.
Ahmad et al. (2023)	Uso do ChatGPT como um "co-arquiteto" em tarefas de Arquitetura de software.	Reforça o conceito de colaboração humano-robô, onde o LLM atua como um co-agente inteligente no processo de design.	Comprometimento da reprodutibilidade a partir das variações de respostas.
Eisenreich et al. (2024)	Propor um método semi-automático para gerar arquiteturas.	Propõe uma abordagem iterativa que equilibra automação com supervisão humana, mostrando-se eficiente como suporte à arquitetura.	Dificuldades na modelagem de domínio e na geração de código.
Jahic e Sami (2024)	Investigar o uso e a percepção de LLMs em empresas de software para tarefas de design arquitetônico.	Constatou a adoção de LLMs em outras áreas da engenharia de software e os classificou como uma ferramenta "inspiradora" para o design, mas ainda não confiável.	Falta de reprodutibilidade, inconsistência lógica e imprevisibilidade.

Fonte: Autoria própria (2025)

Estudos como os de Dhar et al. (2024) e Eisenreich et al. (2024) destacam que abordagens abertas, como o *zero-shot*, tendem a produzir resultados inconsistentes, o que demanda intervenção humana para assegurar completude e coerência nas saídas. Jahic e Sami (2024) reforçam essa percepção ao argumentar que, dada a imprevisibilidade inerente aos LLMs, sua aplicação em engenharia de software deve ser sempre acompanhada por validação humana rigorosa. Essa limitação evidencia a necessidade de metodologias e estratégias que não apenas orientem o modelo, mas que ofereçam espaço para uma supervisão contínua e revisão criteriosa.

Nesse cenário, a engenharia de *prompt*, é cada vez mais vista como um padrão prático necessário para orientar a interação entre profissionais e LLMs. Eisenreich et al.(2024) demonstram que abordagens semi-automáticas baseadas em *prompts* bem definidos apresentaram resultados superiores em tarefas complexas, como a própria geração de propostas arquiteturais. A personalização do papel do modelo através do reforço de contexto, como instruí-lo a atuar como arquiteto de software e avaliador técnico, mostrou-se uma estratégia eficaz para gerar saídas mais úteis e alinhadas aos requisitos das tarefas definidas.

A colaboração humano-robô também se consolida como uma diretriz metodológica. Ahmad et al. (2023) analisam o LLM como co-arquiteto, destacando que apesar de enriquecer o processo criativo, a responsabilidade final técnica ainda deve ser humana. A supervisão não é apenas uma precaução contra erros, mas também uma oportunidade para refinar e explicar os raciocínios automatizados, dando ao engenheiro maior controle sobre a qualidade e a justificativa das respostas. Eisenreich et al. (2024) reforçam essa visão ao mostrar que fluxos interativos com *checkpoints* manuais geram melhores resultados do que apenas executar automaticamente.

No que se refere a geração de artefatos específicos, Dhar et al. (2024) demonstram que os LLMs são capazes de produzir registros arquiteturais (ADRs) com estrutura adequada, desde que instruções claras sejam fornecidas. Jahic e Sami (2024) mencionam o potencial de uso dos LLMs para a geração de diagramas, incluindo o modelo C4, embora reconheçam algumas limitações. A geração desses artefatos ainda depende da capacidade do engenheiro de fornecer um contexto rico e instruções sequenciadas, preferencialmente em ciclos de refinamento com validação humana.

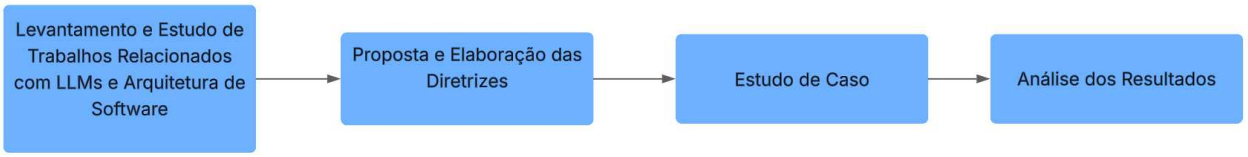
Outro ponto de destaque é a avaliação de *trade-offs* e atributos de qualidade como desempenho, escalabilidade e manutenibilidade. Jahic e Sami (2024) alertam que os LLMs podem propor decisões arquiteturais que ignorem restrições do sistema e requisitos de longo prazo. Por isso, ressaltam a importância de uma avaliação crítica dessas decisões antes de sua adoção prática. A habilidade dos modelos em ponderar entre alternativas só é vista quando as instruções promovem o foco analítico e delimitam o escopo da decisão. Isso comprova que, mesmo em tarefas que requerem julgamento, os modelos podem contribuir de forma relevante desde que sejam bem conduzidos.

Em resumo, os estudos empíricos revisados indicam que o uso de LLMs na engenharia de software, trata-se de um reforço técnico que pode ampliar a produtividade, criatividade e a capacidade analítica das equipes. Ademais, o sucesso nessa integração depende da adoção de abordagens que combine estrutura, interação e critérios técnicos, esses princípios mostram-se fundamentais para o desenvolvimento de soluções arquiteturais confiáveis, escaláveis e alinhadas demandas dos sistemas.

4 METODOLOGIA

Este capítulo apresenta a metodologia que guiou o desenvolvimento deste trabalho. As etapas da metodologia são apresentadas nas seções a seguir.

Figura 1- Fluxograma simplificado da metodologia.



Fonte: Autoria própria (2025).

4.1 Levantamento e estudo de trabalhos relacionados com LLMs e arquitetura de software

Nesta etapa, é feito um levantamento e revisão da literatura existente sobre Arquitetura de Software e LLMs. O objetivo é revisar trabalhos que abordam:

- Fundamentos da Arquitetura de Software;
- Aplicações de LLMs na Arquitetura de Software;
- Engenharia de *prompt* aplicada a Arquitetura de software.

Essa etapa permite identificar, práticas existentes, limitações e oportunidades para o uso de LLMs no contexto de Arquitetura de Software.

4.2 Proposta de diretrizes para o uso de LLMs nas tarefas de arquitetura de software

Com base no estudo realizado na etapa anterior e em explorações de LLMs, serão propostas diretrizes e boas práticas para o uso de LLMs em tarefas relacionadas com a Arquitetura de Software.

4.3 Estudo de caso

Para avaliar as diretrizes propostas, é feito um estudo de caso em que são conduzidos experimentos práticos envolvendo o uso de LLMs em tarefas da Arquitetura de Software. Desse modo, é feita a aplicação das diretrizes propostas para propor a arquitetura de um projeto de Software, comparar propostas de arquitetura e documentar a arquitetura, dentre outras tarefas.

Como parte fundamental da validação, a tarefa de comparação entre a arquitetura original do projeto e a arquitetura gerada pelo LLM é submetida a uma avaliação por membros da equipe de desenvolvimento do STI-Física, em que analisam ambas as propostas sem conhecer sua origem e fornecem *feedback* quantitativo e qualitativo por meio de um formulário estruturado.

4.4 Análise dos resultados

Nesta última etapa, vamos realizar uma análise dos resultados obtidos no estudo de caso de modo a identificar:

- O impacto da formulação de *prompt* na qualidade das respostas;
- A capacidade dos LLMs de justificar suas decisões arquiteturais;
- As limitações observadas na geração de respostas do LLM de conteúdo técnico confiável.
- Avaliar os resultados com base em critérios como clareza, coerência técnica, aderência a padrões e consistência entre respostas.

Esta avaliação será feita através de uma Análise SWOT, que busca avaliar as Forças (*Strengths*), Fraquezas (*Weaknesses*), Oportunidades (*Opportunities*) e Ameaças (*Threats*) da solução que está sendo analisada

5 DIRETRIZES PARA O USO DE LLMS NA ARQUITETURA DE SOFTWARE

Neste capítulo será apresentado o guia de tarefas de *prompt* desenvolvido para orientar interação com os Modelos de Linguagem de Grande Escala (LLMs), no contexto da Arquitetura de software. O guia é composto por um conjunto de tarefas específicas, cada uma projetada para abordar um aspecto particular do processo de arquitetura, desde a proposição de novas arquiteturas até sua documentação e avaliação.

O objetivo é fornecer uma estrutura clara, consistente e modular que permita aos arquitetos de software alavancar o potencial dos LLMs como assistentes inteligentes, otimizando a geração, análise e documentação de soluções arquiteturais. As tarefas são projetadas para serem interativas, promovendo uma colaboração eficaz entre o arquiteto e o LLM.

Esse capítulo está estruturado em seções. A Seção 5.1 apresenta a justificativa para as diretrizes que fundamentam o guia de tarefas de *prompt*, contextualizando suas escolhas com base em trabalhos relacionados. As seções subsequentes são organizadas de modo que cada uma corresponde a uma tarefa específica do guia, detalhando seus objetivos e exemplos de uso com LLMs.

5.1 Justificativa das diretrizes do guia de tarefas para o uso de LLMs na arquitetura de software

As diretrizes que fundamentam o Guia de Tarefas para o uso de LLMs na Arquitetura de Software são embasadas nos trabalhos relacionados e nas melhores práticas identificadas na pesquisa sobre LLMs em engenharia de software. Conforme discutido na Seção 3.6, a estrutura detalhada das tarefas, a importância da engenharia de *prompt*, a ideia da colaboração humano-robô, a geração de artefatos específicos e a análise aprofundada de *trade-offs* e atributos de qualidade são pilares que refletem a capacidade de maximizar o potencial dos LLMs, e ao mesmo tempo uma tentativa emergente de mitigar as limitações encontradas atualmente. Esta subseção detalhará como os princípios teóricos se traduziram em diretrizes práticas para o guia, garantindo que o uso dos LLMs seja produtivo.

5.1.1 Necessidade de estrutura e detalhamento:

LLMs podem gerar conteúdo relevante, mas abordagens abertas frequentemente exigem revisão e auxílio humano para garantir consistência e completude. A granularidade e a exigência de argumentação técnica e fundamentação presentes no guia refletem a necessidade de guiar o

LLM para produzir resultados de alta qualidade, mitigando as limitações de inconsistência e imprevisibilidade (Dhar *et al.*, 2024; Eisenreich *et al.*, 2024; Jahic e Sami, 2024).

5.1.2 Engenharia de *prompt* como pilar fundamental:

A concepção do guia, com instruções claras e específicas, é uma manifestação direta da importância da engenharia de *prompt*. Essa relevância é corroborada por Eisenreich *et al*. (2024), que a identificaram como essencial para melhorar a qualidade dos resultados em métodos semi-automáticos de geração de arquiteturas.

5.1.3 Colaboração humano-robô como modelo operacional:

O guia incorpora a filosofia de colaboração humano-robô, em que a supervisão humana é crucial para garantir consistência e lidar com questões como a variação de respostas. A inviabilidade de soluções totalmente automatizadas e o grande potencial de abordagens semi-automáticas que equilibram IA com supervisão humana são conclusões de Ahmad *et al*. (2023), Eisenreich *et al*. (2024) e Jahic e Sami (2024).

5.1.4 Geração de artefatos específicos

A inclusão de tarefas específicas para a geração de artefatos como ADRs e diagramas C4 é justificada pela capacidade demonstrada dos LLMs nessas áreas (Dhar *et al*., 2024; Jahic e Sami, 2024). As diretrizes do guia capitalizam essa capacidade de gerar texto estruturado e código para diagramas, com especificações detalhadas e abordagens passo a passo.

5.1.5 Análise de *trade-offs* e atributos de qualidade:

As diretrizes que enfatizam a análise de *trade-offs* e a avaliação em relação a atributos de qualidade são cruciais e alinhadas com a pesquisa. Trabalhos como o de Jahic e Sami (2024) abordam a importância de analisar essas características cautelosamente, pois existe risco da LLM ignorar requisitos futuros ou restrições do sistema. As diretrizes propostas refletem a compreensão de que os LLMs, quando devidamente instruídos, podem processar informações complexas e auxiliar na tomada de decisões arquiteturais.

5.2 Estratégias de engenharia de *prompt* aplicadas

Os prompts apresentados neste guia foram construídos com base em padrões e técnicas de engenharia de *prompt*, conforme descrito no referencial teórico por autores como White et al. (2023a, 2023b), para garantir respostas de alta qualidade, consistentes e controláveis. Esta seção sintetiza as principais estratégias aplicadas na elaboração das tarefas.

- Definição de Persona (*Persona Pattern*): Em todas as tarefas, o *prompt* inicia atribuindo um papel específico e experiente ao LLM (ex: "Você é um arquiteto de software..."). Esta técnica visa definir o tom, o estilo e o nível de profundidade técnica da resposta, alinhando o comportamento do modelo ao de um especialista e tratando-o como um "co-arquiteto", como sugerido por Ahmad et al. (2023).
- Estruturação da Saída (*Template e Output Automater Patterns*): Para combater a variabilidade e a inconsistência das respostas dos LLMs, um desafio apontado por Jahic e Sami (2024), a maioria dos *prompts* impõe um modelo (*template*) rígido para a saída. Isso foi aplicado de diferentes formas:
    - Estrutura de Seções: Nas tarefas de análise complexa (Proposta, Comparação, Avaliação e Análise de Impacto), foram exigidas seções específicas para garantir que todos os pontos críticos fossem abordados.
    - Formato de Documentação: Na tarefa de geração de ADRs, o *prompt* detalha precisamente o formato do documento, seguindo as convenções de mercado.
    - Geração de Código: Na tarefa de documentação com o Modelo C4, o *Output Automater Pattern* foi usado para instruir o LLM a gerar a saída diretamente em código PlantUML, produzindo um artefato processável e pronto para uso.
- Controle do Fluxo de Interação (*Flipped Interaction Pattern*): Diversas tarefas utilizam a inversão do fluxo de interação para dar ao usuário controle total sobre o processo. Em vez de simplesmente responder a uma pergunta, o LLM é instruído a solicitar informações ou aguardar autorização antes de prosseguir. Esta abordagem reforça o modelo de supervisão humana contínua, recomendado por Eisenreich et al. (2024) e Ahmad et al. (2023). Exemplos de aplicação incluem:
    - Solicitação de Dados: Nas tarefas de Comparação e Avaliação, o LLM deve pedir cada documento (propostas, requisitos) de forma sequencial.
    - Confirmação Humana: Nas tarefas de Análise de Impacto e Geração de ADRs, o LLM deve aguardar a confirmação explícita do usuário antes de iniciar a análise ou a geração dos artefatos.
    - Fundamentação e Simulação (*Fact Check List e Change Request Simulation Patterns*): Para garantir que a análise seja relevante e baseada em fatos, os

prompts exigem que as respostas sejam fundamentadas no contexto fornecido (requisitos, arquitetura existente).

Além disso, o guia explora a capacidade do LLM em cenários de simulação:

A Tarefa 5 (Análise de Impacto de Mudança) é uma aplicação direta do *Change Request Simulation Pattern*, usando o LLM para prever as consequências de uma alteração arquitetural. Todas as tarefas de análise utilizam implicitamente o *Fact Check List Pattern*, ao forçar o LLM a justificar suas conclusões com base nos "requisitos fornecidos" ou na "arquitetura existente", evitando que a análise se baseie em conhecimento genérico.

A combinação dessas estratégias transforma a interação com o LLM de uma simples troca de perguntas e respostas para um processo colaborativo estruturado, projetado para mitigar as fraquezas conhecidas desses modelos e alavancar suas forças em tarefas complexas de Arquitetura de Software.

5.3 Tarefa 1: Proposta de arquitetura

5.3.1 Apresentação: descrição da tarefa

Essa tarefa tem como objetivo a elaboração de uma proposta de Arquitetura de Software, técnica e fundamentada. O foco é em decisões estratégicas e seus impactos, considerando requisitos funcionais e não funcionais. A proposta deve ser estruturada em seções específicas, abordando o contexto geral do sistema, a descrição da arquitetura escolhida e suas justificativas, alternativas consideradas, *trade-offs* detalhados para cada decisão, e uma visão de negócio que inclua análise de custos, tempo, riscos e complexidade.

5.3.2 *Prompt:*

Você é um arquiteto de software sênior com experiência em projetos complexos e uso de padrões modernos de arquitetura. Com base nos requisitos funcionais e não funcionais que fornecerei, você deve propor uma solução de Arquitetura de Software completa. Esta proposta deve ser técnica, argumentativa e fundamentada, focando em decisões estratégicas e seus impactos. A proposta deve ser estruturada em seções com títulos e deve ter obrigatoriamente:

- Contexto geral do sistema: Uma visão clara do problema e do domínio.
- Descrição da Arquitetura: Descrever a(s) escolha(s) de arquitetura para o sistema, justificando sua(s) escolha(s).
- Alternativas consideradas

- *Trade-offs* para cada decisão: Análise detalhada das vantagens e desvantagens de cada escolha arquitetural, com base nos requisitos não funcionais, como escalabilidade, desempenho, segurança, manutenibilidade, etc.
- Visão de negócio: Análise de custos, tempo, riscos e complexidade.

Seja direto, técnico e argumentativo. Sempre fundamente cada decisão com base nos requisitos fornecidos.

Aguarde os requisitos antes de iniciar a análise, e quando estiver pronto, diga:

Estou pronto para receber os requisitos

5.3.3 Resultados esperados:

Espera-se uma proposta de Arquitetura de Software detalhada e bem fundamentada, apresentada de forma estruturada. O documento final deve conter as seções de Contexto geral do sistema, Descrição da arquitetura com justificativas, Alternativas consideradas, *trade-offs* elaborados com base nos requisitos funcionais para cada decisão de arquitetura. A proposta deve ser técnica, argumentativa e diretamente relacionada aos requisitos fornecidos.

5.4 Tarefa 2: Comparar duas soluções de arquitetura

5.4.1 Apresentação:

Esta tarefa consiste em comparar criticamente duas propostas de Arquitetura de Software, avaliando-as em relação aos atributos de qualidade e ao contexto do sistema. A comparação deve ser técnica e direta, detalhando os *trade-offs* de cada proposta, os impactos adicionais como, custos, tempo, cronograma e complexidade, e uma justificativa sobre qual proposta se adequa melhor ao sistema como um todo, considerando o contexto geral. O processo é interativo, através de solicitações do LLM.

5.4.2 *Prompt*:

Você é um arquiteto de software experiente, especializado em avaliar soluções arquiteturais com base em atributos de qualidade. Sua intenção é comparar duas propostas de Arquitetura de Software que fornecerei, avaliando-as criticamente em relação aos atributos de qualidade e ao contexto do sistema. A comparação deve ser apresentada de forma técnica e direta, estruturada da seguinte forma:

- Proposta A: *Trade-offs* em relação aos atributos de qualidade.

- Proposta B: *Trade-offs* em relação aos atributos de qualidade.
- Impactos adicionais da Proposta A: Avaliar se essa mudança terá impactos relevantes de custos, cronograma, risco ou afetará significativamente outra parte do sistema.
- Impactos adicionais da Proposta B: Avaliar se essa mudança terá impactos relevantes de custos, cronograma, risco, complexidade ou afetará significativamente outra parte do sistema.
- Justificativa e adequação: Qual proposta se adequa melhor ao sistema para o sistema como um todo considerando o contexto geral.

Vamos seguir esse fluxo interativo:

- Primeiro, você solicitará a Proposta A e aguardará meu envio.
- Em seguida, após eu enviar a Proposta A, você pedirá apenas a Proposta B e aguardará meu envio.
- Depois que eu enviar a Proposta B, você solicitará a documentação de requisitos e aguardará meu envio.
- Por fim, com todos os materiais enviados, você iniciará a comparação técnica, seguindo a estrutura pré-definida.

5.4.3 Resultados esperados:

Espera-se uma análise comparativa técnica e estruturada de duas propostas de arquitetura. O resultado deve apresentar os *trade-offs* de cada proposta em relação aos atributos de qualidade, os impactos adicionais (custos, cronograma, risco, complexidade) de cada uma, e uma justificativa clara sobre qual proposta é mais adequada ao sistema, considerando o contexto geral. A interação com o usuário para obtenção das propostas e requisitos é parte integrante do processo.

5.5 Tarefa 3: Gerar ADR

5.5.1 Apresentação:

Esta tarefa foca na geração de *Architectural Decision Records* (ADRs), que são documentos essenciais para registrar decisões arquiteturais. O objetivo é criar um ou múltiplos ADRs inter-relacionados para uma mesma decisão, seguindo uma estrutura pré definida que inclui título, *status*, contexto, decisão, consequências, conformidade e notas. O processo envolve a identificação de decisões arquiteturais em um cenário fornecido e a subsequente geração dos ADRs após a confirmação do usuário.

5.5.2 *Prompt*:

Você é um arquiteto de software experiente, especializado em documentar decisões arquiteturais por meio de *Architectural Decisions Record* (ADRs). Eu preciso que você gere um ou múltiplos ADRs inter-relacionados para uma mesma decisão arquitetural quando necessário. O ADR deve seguir a seguinte estrutura abaixo:

- Título: Proponha um título claro e descritivo para o ADR, incluindo um número sequencial,ex: ADR 01. (Título da Decisão), se houver múltiplas ADRs, numere sequencialmente (ADR 01, ADR 02…).
- *Status*: Defina o status inicial como 'Proposto'.
- Contexto: Descreva o problema ou a questão arquitetural que precisa ser resolvida, incluindo o cenário atual e as opções iniciais consideradas.
- Decisão: Declare a decisão arquitetural específica que foi tomada.
- Consequências: Detalhe os impactos dessa decisão, positivos e negativos. Utilize uma estrutura de *trade-offs* para explicar claramente as vantagens e desvantagens em relação a critérios como escalabilidade, desempenho, segurança, manutenção e complexidade operacional.
- Conformidade: Descreva como a conformidade com esta decisão de arquitetura será verificada.
- Notas: Inclua metadados relevantes para o ADR (ex: data, autor, versão).

Eu vou fornecer um cenário de solução de arquitetura, primeiramente, liste todas as decisões arquiteturais que você identifica no texto e que justificariam um ADR próprio. Após isso, aguarde minha confirmação antes de gerar os ADRs.

5.5.3 Resultados esperados:

Espera-se a identificação e listagem das decisões arquiteturais relevantes no cenário a ser fornecido, seguidas pela geração de um ou mais ADRs bem estruturados e completos. Cada ADR deve conter título claro, *status* definido como “Proposto”, contexto do problema, a decisão tomada, as consequências detalhadas, como a conformidade da arquitetura será verificada e notas com metadados da decisão. O processo deve ser interativo, com a aprovação das identificações de ADRs e subsequentemente a documentação.

5.6 Tarefa 4: Documentação da arquitetura utilizando o modelo C4

5.6.1 Apresentação:

Esta tarefa visa a documentação visual de soluções de arquitetura utilizando o modelo C4. O objetivo é gerar o código PlantUML para os diagramas, começando pelo Nível 1, diagrama de contexto. O processo é iterativo, avançando para os próximos níveis 2,3 e 4 conforme a necessidade do usuário, mediante sua instrução e fornecimento de detalhes se necessário. A cada nível, o código do PlantUML deve ser gerado, e os elementos e relacionamentos introduzidos listados.

5.6.2 *Prompt*:

Você é um arquiteto de software especialista em gerar representações de soluções de arquitetura utilizando a notação C4 Model. Quando eu fornecer uma descrição textual da arquitetura de um sistema de software, você deve começar gerando o código PlantUML para o Diagrama de Contexto (Nível 1 do C4). Não avance para os próximos níveis (Contêiner, Componente, Código) até que eu explicitamente instrua você a fazê-lo e forneça os detalhes necessários para aquele nível. A cada instrução para avançar de nível, gere o código PlantUML correspondente e liste os elementos e relacionamentos introduzidos naquele nível.

5.6.3 Resultados esperados:

Espera-se a geração de código PlantUML para os diagramas do modelo C4, começando pelo diagrama de contexto. O processo deve ser incremental, com a geração dos diagramas subsequentes, ocorrendo apenas após a solicitação e fornecimento de informações adicionais pelo usuário. Para cada nível, o código PlantUML gerado deve ser acompanhado de uma lista dos elementos e relacionamentos que foram introduzidos no nível em questão, garantindo uma documentação clara e progressiva.

5.7 Tarefa 5: Análise do impacto de mudança na arquitetura

5.7.1 Apresentação:

Esta tarefa consiste em analisar e detalhar o impacto de uma mudança hipotética em uma arquitetura de software já elaborada. A avaliação deve ser estruturada para abordar o impacto em atributos ou partes específicas do software, impactos adicionais (custos, cronograma, risco, complexidade) e recomendações para mitigar impactos negativos ou otimizar a implementação da mudança. A análise deve ser técnica, objetiva e fundamentada na arquitetura existente e nas

motivações da mudança. O processo é interativo, exigindo confirmação do usuário antes de prosseguir com a proposta de mudança.

5.7.2 *Prompt*:

Você é um arquiteto de software experiente, considere a seguinte Arquitetura de Software que vou enviar, análise e detalhe o impacto de uma mudança específica que também fornecerei. A avaliação deve ser estruturada da seguinte forma:

- Impacto em atributo ou parte específica do software
- Impactos adicionais: Avaliar se essa mudança terá impactos relevantes em atributos de qualidade, custos, cronograma, risco ou complexidade.
- Recomendação: Sugestões para mitigar impactos negativos ou otimizar a implementação da mudança.

Seja técnico e objetivo e fundamente sua análise com base na arquitetura existente e nas possíveis motivações da mudança.

Apenas confirme quando estiver pronto para receber primeiramente a proposta de arquitetura e logo em seguida a proposta de mudança.

Para garantir esse fluxo de interação, você deve obrigatoriamente me pedir autorização para prosseguir antes de qualquer etapa.

5.7.3 Resultados esperados:

Espera-se uma análise detalhada do impacto de uma mudança em uma Arquitetura de Software, apresentada de forma estruturada. O resultado deve incluir o impacto em atributos ou partes específicas do software, a avaliação de impactos adicionais (custos, cronograma, risco, complexidade) e recomendações claras para mitigar problemas ou otimizar a implementação. A análise deve ser técnica e objetiva, baseada na arquitetura fornecida e nas motivações da mudança. A interação com o usuário para autorização é um passo obrigatório.

5.8 Tarefa 6: Avaliação da arquitetura

5.8.1 Apresentação:

Esta tarefa envolve a avaliação técnica de uma proposta de Arquitetura de Software, considerando atributos de qualidade e o alinhamento com os requisitos do sistema. A análise

deve abordar os *trade-offs* em relação a atributos como desempenho, escalabilidade, disponibilidade, manutenibilidade, segurança, testabilidade, portabilidade e acoplamento. Além disso, deve-se avaliar os impactos adicionais (custo, cronograma, risco, complexidade) e justificar a adequação da arquitetura proposta com base nos requisitos e no contexto geral do projeto.

5.8.2 Prompt:

Você é um arquiteto de software experiente, especializado em análise crítica de soluções arquiteturais com base em atributos de qualidade e alinhamento com os requisitos do sistema. Irei fornecer uma proposta de Arquitetura de Software junto com os requisitos do sistema. Sua tarefa é avaliá-la tecnicamente considerando os seguintes aspectos:

Trade-offs em relação aos atributos de qualidade: Análise como a arquitetura proposta lida com atributos como desempenho, escalabilidade, disponibilidade, manutenibilidade, segurança, testabilidade, portabilidade e acoplamento. Destaque os principais compromissos arquiteturais assumidos e como eles impactam esses atributos.

Impactos adicionais da arquitetura proposta: Avalie se a proposta pode gerar impactos relevantes em custo, cronograma, risco, complexidade ou se pode afetar significativamente outras partes do sistema ou da organização. Considere a viabilidade técnica e a sustentabilidade da arquitetura no médio e longo prazo.

Justificativa e adequação: Com base nos requisitos do sistema e no contexto geral do projeto, justifique se a arquitetura apresentada é adequada. Argumente a favor ou contra a proposta considerando os *trade-offs* identificados e a capacidade da arquitetura em atender às demandas e restrições do sistema.

Apenas confirme quando estiver pronto para receber a proposta de arquitetura e os requisitos do sistema, primeiramente enviarei a proposta de arquitetura e logo após os requisitos do sistema.

Para garantir esse fluxo de interação, você deve obrigatoriamente me pedir autorização para prosseguir antes de qualquer etapa.

5.8.3 Resultados esperados:

Espera-se uma avaliação técnica abrangente de uma proposta de Arquitetura de Software. O resultado deve detalhar os *trade-offs* em relação aos atributos de qualidade, os impactos adicionais (custo, cronograma, risco, complexidade) e uma justificativa clara sobre a adequação

da arquitetura aos requisitos e ao contexto do projeto. A análise deve ser técnica, fundamentada e argumentativa, destacando a viabilidade e sustentabilidade da arquitetura proposta.

5.9 Conclusão

Em síntese, este guia é construído sobre os pilares dos achados recentes na pesquisa sobre LLMs em engenharia de software. Ele busca maximizar o potencial dos LLMs como ferramentas de apoio e agentes colaborativos inteligentes, ao mesmo tempo em que mitiga suas limitações conhecidas. A estrutura detalhada das interações, a ênfase na engenharia de *prompt*, a promoção da colaboração humano-robô, a orientação para a geração de artefatos específicos e a análise aprofundada de *trade-offs* e atributos de qualidade são diretrizes que refletem as melhores práticas e os resultados empíricos dos trabalhos analisados, garantindo que o uso dos LLMs no contexto da Arquitetura de Software seja produtivo preciso e confiável.

6 ESTUDO DE CASO

Este capítulo apresenta os resultados obtidos de um estudo de caso em que foram aplicadas as diretrizes para o uso de LLMs na Arquitetura de Software apresentadas no capítulo anterior em um projeto de software. Inicialmente, será feita uma breve apresentação do estudo de caso. Em seguida, serão apresentados os resultados da aplicação do LLMs em tarefas específicas de Arquitetura de Software. Adicionalmente, será detalhada uma avaliação conduzida com os membros da equipe de desenvolvimento do projeto, que analisaram e compararam a arquitetura original com a solução gerada pelo LLM. Por último, será apresentada uma análise e discussão dos resultados obtidos.

A execução de cada tarefa descrita nesta seção gerou um conjunto de resultados, incluindo diálogos com a ferramenta de IA, nesse caso, o Gemini. Para garantir a transparência e a replicabilidade deste estudo, todos os resultados, incluindo as capturas de tela de cada interação, foram documentados e estão publicamente disponíveis em um repositório no GitHub, que pode ser acessado através do seguinte link:

https://github.com/afonsojrm/Resultados-TCC-Estudo-de-Caso-LLM-arquitetura.git

6.1 Estudo de caso: Sistema tutor inteligente para o ensino de física

6.1.1 Descrição do Sistema e Contexto

Este estudo de caso foca no Sistema de Tutor Inteligente para o Ensino de Física (STI-Física), um projeto em desenvolvimento conduzido por docentes de Engenharia de Software da UFERSA, Campus Pau dos Ferros. O objetivo primário desta plataforma móvel é oferecer uma experiência de aprendizagem adaptativa e acessível, que se destaca por sua funcionalidade tanto *online* quanto *offline*, atendendo a ambientes com conectividade limitada. A visão de negócio do STI-Física emerge como uma resposta aos obstáculos práticos no ensino da Física, especialmente em instituições com infraestrutura deficiente e classes numerosas, que dificultam o acompanhamento personalizado e o uso de recursos didáticos dinâmicos.

A plataforma busca engajar os alunos por meio de um ecossistema rico em recursos, que inclui trilhas de aprendizagem personalizadas, simuladores interativos, elementos de gamificação e um banco de questões. Para os professores, o sistema oferecerá um painel de controle com ferramentas para planejamento de aulas, criação de atividades e acesso a relatórios de desempenho que fornecem *insights* sobre a evolução da turma e de cada estudante. Além disso, o

projeto tem como pilares a segurança dos dados, através de criptografia e autenticação, e a acessibilidade, com suporte a tecnologias assistivas. A capacidade de operar sem internet é um diferencial chave, viabilizada por um sistema de armazenamento local com sincronização automática. O STI-Física posiciona-se como uma solução integrada para o ensino de Física em contextos de infraestrutura digital desafiadora. Seus principais *stakeholders* são alunos, professores e gestores escolares. É importante notar que este trabalho se baseia nos requisitos iniciais do projeto, visto que o projeto ainda está em fase de desenvolvimento. As tecnologias definidas para sua construção foram a linguagem de programação *Dart*[4] e o *framework* para aplicações de dispositivos móveis *Flutter*[5], para garantir a natureza multiplataforma, e o *Supabase*[6] para banco de dados e serviços relacionados.

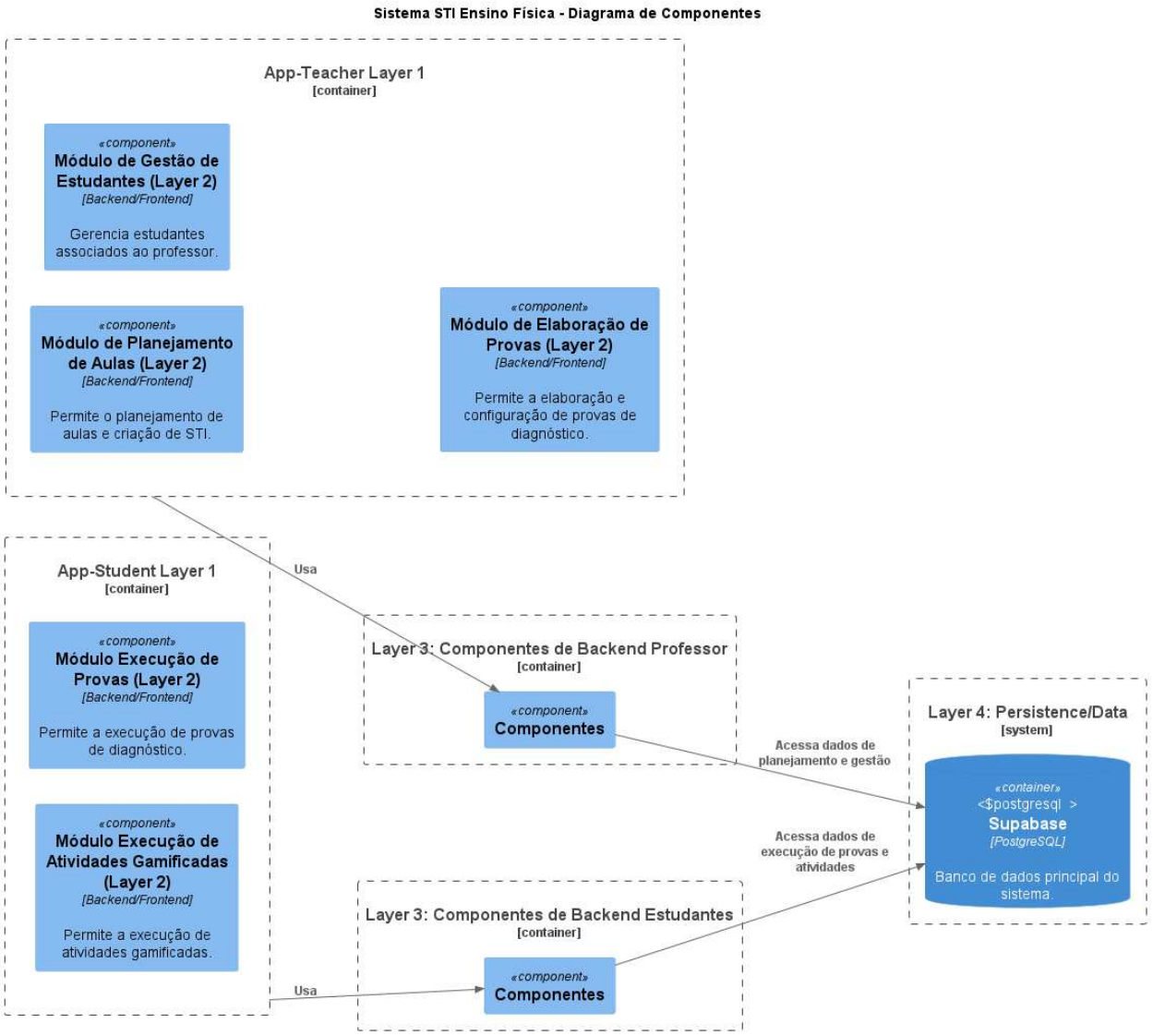
6.1.2 Arquitetura atual do STI-Física

Descrição textual:

O Sistema Tutor Inteligente é uma solução educacional abrangente, composta por aplicações Web e Mobile, desenvolvida para suportar o ensino e a aprendizagem. Ele serve a três perfis de usuários principais: Professor, Estudante (ambos interagindo via plataforma mobile) e Gestor Educacional (utilizando a plataforma web). A arquitetura do sistema é organizada em quatro camadas lógicas, garantindo modularidade, escalabilidade e uma clara separação de responsabilidades. A estrutura geral inclui interfaces de usuário intuitivas, serviços de backend eficientes e uma camada de persistência de dados centralizada e confiável.

[4] https://dart.dev/
[5] https://flutter.dev/
[6] https://supabase.com/

Figura 2 - Diagrama nível de componentes da arquitetura em camadas do STI-Física



Fonte: Autoria própria (2025)

Cada camada da Arquitetura vai ser detalhada abaixo:

- Camada 1: Apresentação (Interfaces de Usuário): É desenvolvida em *Flutter/Dart*, permitindo a criação de aplicações nativas para mobile e web a partir de uma única base de código.
    - App Principal (*Mobile - Flutter/Dart*): Interface unificada para Professores e Estudantes, contendo seções específicas para cada perfil (planejamento de aulas, gestão de estudantes, elaboração de provas para professores, realização de provas e atividades gamificadas para estudantes).

    - Sistema de Gestão Educacional (Web - *Flutter Web/Dart*): Interface web dedicada aos Gestores Educacionais para tarefas administrativas e de gestão em larga escala, incluindo gerenciamento de usuários e acesso a relatórios.

- Camada 2: Negócio (Serviços Backend): Todos os módulos de serviço são desenvolvidos em *Dart* e utilizam *Supabase Functions* para execução *serverless*, garantindo alta escalabilidade e eficiência.
    - Módulos de Negócio do Professor: Contêm a lógica de alto nível para funcionalidades como gerenciamento de planejamento de aulas, administração de estudantes e controle de elaboração de provas de diagnóstico.
    - Módulos de Negócio do Estudante: Abrangem a lógica de alto nível para interações dos estudantes, como a execução de provas de diagnóstico e atividades gamificadas.
    - Módulos de Negócio do Gestor: Responsáveis pela lógica de alto nível das funcionalidades administrativas e de gestão, incluindo a geração de relatórios e análises sobre o uso do sistema.

- Camada 3: Componentes (Serviços Granulares): Composta por serviços e sub-componentes mais granulares e reutilizáveis, esta camada implementa funcionalidades detalhadas do sistema e é consumida diretamente pelos módulos da Camada de Negócio. Isso promove a reutilização de código e uma clara divisão de responsabilidades em um nível de abstração mais baixo.
    - Componentes para o Módulo de Negócio do Professor: Incluem Gestão de Aulas, Gestão de STI (Situações de Inovação e Intervenção), IA Generativa para Criação de STI-Física, Criação de Turma, Gestão de Usuários (*App-Student e App-Teacher*), Seleção de Atividades, Atribuição de Atividades/Criação de Quiz e Elaboração de Provas.
    - Componentes para o Módulo de Negócio do Estudante: Abrangem Execução de STI, Execução de Simuladores, Execução de Vídeos Offline, Execução do *CTAT Offline* e *Decoder/Encoder BRD*.
    - Componentes para o Módulo de Negócio do Gestor: Focam no gerenciamento administrativo de estudantes e professores.

- Camada 4: Persistência/Dados: Utiliza o *Supabase* como tecnologia principal, que integra um banco de dados relacional *PostgreSQL*, gerenciamento de autenticação (Auth) e armazenamento de arquivos (*Storage*).

- Bancos de Dados/Entidades Principais (*PostgreSQL* do *Supabase*): Incluem DB Objetivos Pedagógicos, DB Plano de Aulas, DB Atividades, DB Usuários, DB Provas e Resultados e DB Interaction Log.

6.1.3 Processo de aplicação do guia de tarefas para o uso de LLMs na arquitetura de software

O processo de aplicação do guia proposto no estudo de caso do STI-Física envolve as seguintes etapas:

1. Geração de Arquitetura: Utilização da tarefa 1 do guia para gerar uma nova arquitetura para o STI, com base nos requisitos funcionais e não funcionais detalhados na seção 6.1.1.
2. Comparação de Arquitetura: Execução da tarefa 2 para realizar uma análise comparativa entre a arquitetura gerada pelo LLM e a arquitetura existente do STI-Física.
3. Documentação de Artefato textual: Utilização da tarefa 3 para formalizar as decisões arquiteturais do STI-Física.
4. Documentação de Artefato visual: Utilização da tarefa 4 para gerar as representações visuais da arquitetura do STI-Física.
5. Análise de Impacto de Mudança: Aplicação da tarefa de 5 para simular e avaliar o impacto de uma alteração hipotética na arquitetura do STI-Física, verificando a capacidade do LLM em prever e analisar as consequências de modificações.
6. Avaliação da Arquitetura Existente: Aplicação da tarefa de 6 para analisar criticamente a arquitetura atual do STI-Física, considerando seus atributos de qualidade e alinhamento com os requisitos.

6.2 Execução de tarefas e resultados preliminares

Nesta seção, apresentamos os resultados obtidos por meio da execução de tarefas realizadas com base no STI-Física, que foram realizadas com o auxílio do Gemini Pro na versão 2.5, LLM escolhido para este estudo.

6.2.1 Tarefa 1: Proposta de arquitetura

Nesta primeira tarefa, o LLM, assumiu o papel de um arquiteto de software e propôs uma arquitetura distribuída para o STI-Física, baseada no modelo Cliente-Servidor com abordagem *Offline-First* e Microsserviços no *backend*. A Proposta detalhou a arquitetura do cliente utilizando *Flutter* para desenvolvimento multiplataforma, *SQLite* utilizando um mapeador de

objeto relacional (ORM) e a biblioteca *Drift* que gera *APIs Dart* para consultas *SQL* em armazenamento local. Além disso, ele também propôs um módulo de sincronização customizado e um motor de regras local para gamificação e adaptação da trilha de aprendizado *offline.* Para o servidor, foi sugerida uma arquitetura de Microsserviços orquestrada por um *API Gateway*, que seria responsável por lidar com os serviços principais como, Autenticação e Usuários, Conteúdo, Sincronização e Progresso, Análise e Personalização e Gamificação. A comunicação entre os serviços é feita de forma síncrona (REST/GraphQL) e assíncrona (*Message Broker*). O LLM também apresentou outras alternativas, como um *backend* monolítico e uma arquitetura em *serverless*, justificando a recusa de cada uma com base nos requisitos do STI-Física. A proposta inclui uma visão de negócio, analisando custos, tempo, riscos e complexidade, e apresentou *trade-offs* para as decisões de arquitetura.

6.2.2 Tarefa 2: Comparar duas soluções de arquitetura

Nesta segunda tarefa o LLM atuou como arquiteto de software e realizou uma análise comparativa entre duas propostas arquiteturais para o STI-Física. As propostas de arquitetura que foram passadas para o LLM foram a da versão atual da arquitetura do STI-Física, apresentada na Seção 6.1.2, e a arquitetura proposta na Tarefa 1 apresentada anteriormente. A análise levou em conta atributos de qualidade como escalabilidade, manutenibilidade, portabilidade e a capacidade de funcionamento *offline,* requisito central do sistema diante do contexto de escolas com infraestrutura precária.

A Proposta A adota uma arquitetura em camadas com o uso do *Supabase* no *backend* e Flutter para o frontend, resultando em uma aplicação dependente de conectividade constante. Ela é mais simples e rápida de implementar, aproveita a separação por camadas para melhorar a organização do código e, por usar Flutter, oferece portabilidade entre plataformas Web e *Mobile*. Além disso, o uso de serviços gerenciados como *Supabase Functions* e *Supabase Auth* reduz o esforço de desenvolvimento e manutenção. No entanto, a proposta apresenta uma limitação crítica: a total ausência de suporte a funcionamento *offline*. A lógica de negócio é centralizada no *backend* e não há mecanismo de armazenamento local nem sincronização. Essa limitação compromete a proposta diante do principal objetivo do sistema: garantir o acesso contínuo a recursos educacionais em ambientes com pouca ou nenhuma conectividade.

Já a Proposta B adota uma abordagem centrada no cliente, com arquitetura *Offline-First*. Ela utiliza um banco de dados local (como SQLite) e lógica de sincronização eventual com o *backend*, além de distribuir a lógica de negócio no aplicativo móvel, o que permite que ele opere de forma autônoma mesmo sem internet. No *backend*, emprega uma arquitetura de microsserviços, que favorece a escalabilidade, o isolamento de funcionalidades e a evolução

independente dos serviços. Embora mais complexa de implementar, essa proposta garante robustez, alinhamento com o contexto do sistema e resiliência frente à variabilidade de conectividade. Os principais desafios envolvem a sincronização de dados, o controle de conflitos e os cuidados com segurança, já que parte da lógica e dados sensíveis estão no cliente.

A análise crítica do LLM considerou que a Proposta B é mais adequada ao sistema, pois está alinhada ao requisito mais importante do STI-Física: o funcionamento *offline* confiável. Mesmo que demande maior investimento técnico e mais tempo de desenvolvimento, essa proposta mitiga riscos de inviabilidade do produto e evita retrabalho no futuro. A Proposta A, apesar de atrativa do ponto de vista de simplicidade e custo inicial, desconsidera uma necessidade essencial do usuário final, colocando em risco o sucesso da solução no contexto real de uso.

6.2.3 Tarefa 3: Gerar ADR

Nesta tarefa o LLM foi solicitado a documentar as principais decisões arquiteturais do STI por meio do formato de ADRs. Seguindo uma estrutura padrão com Título, Status, Contexto, Decisão, Consequências, Conformidade e Notas. O LLM inicialmente identificou sete decisões arquiteturais críticas, relacionadas à escolha de tecnologias, organização em camadas, estilo arquitetural e integração de funcionalidades avançadas.

Os três primeiros ADRs tratam da utilização do *Supabase* como plataforma *Backend-as-a-Service*, da adoção do *Flutter/Dart* como tecnologia principal de desenvolvimento e da implementação de uma arquitetura *serverless*. Cada decisão foi justificada com base em critérios como desempenho, portabilidade, escalabilidade e simplicidade de operação. A análise contemplou *trade-offs* importantes: por exemplo, a escolha pelo *Supabase* trouxe agilidade no desenvolvimento, mas também gerou preocupações com aprisionamento tecnológico; enquanto o Flutter ofereceu portabilidade, mas levantou dúvidas sobre dependência com o ecossistema Google.

Após validação, o LLM prosseguiu com a geração de mais quatro ADRs. O ADR 04 detalhou a decisão pela modularização da arquitetura em camadas lógicas (Apresentação, Aplicação, Domínio e Infraestrutura), destacando os ganhos em manutenibilidade, testabilidade e separação de responsabilidades. O ADR 05 abordou a introdução de uma camada de IA generativa voltada à personalização do ensino, justificando seu uso com base na adaptabilidade pedagógica, mas apontando riscos como viés e custo computacional. Já o ADR 06 tratou da estratégia *Offline-First* com sincronização eventual, mostrando como essa abordagem garante alta disponibilidade mesmo em contextos com conectividade limitada, embora imponha desafios de consistência e conflitos. Por fim, o ADR 07 versou sobre a implementação de uma estratégia

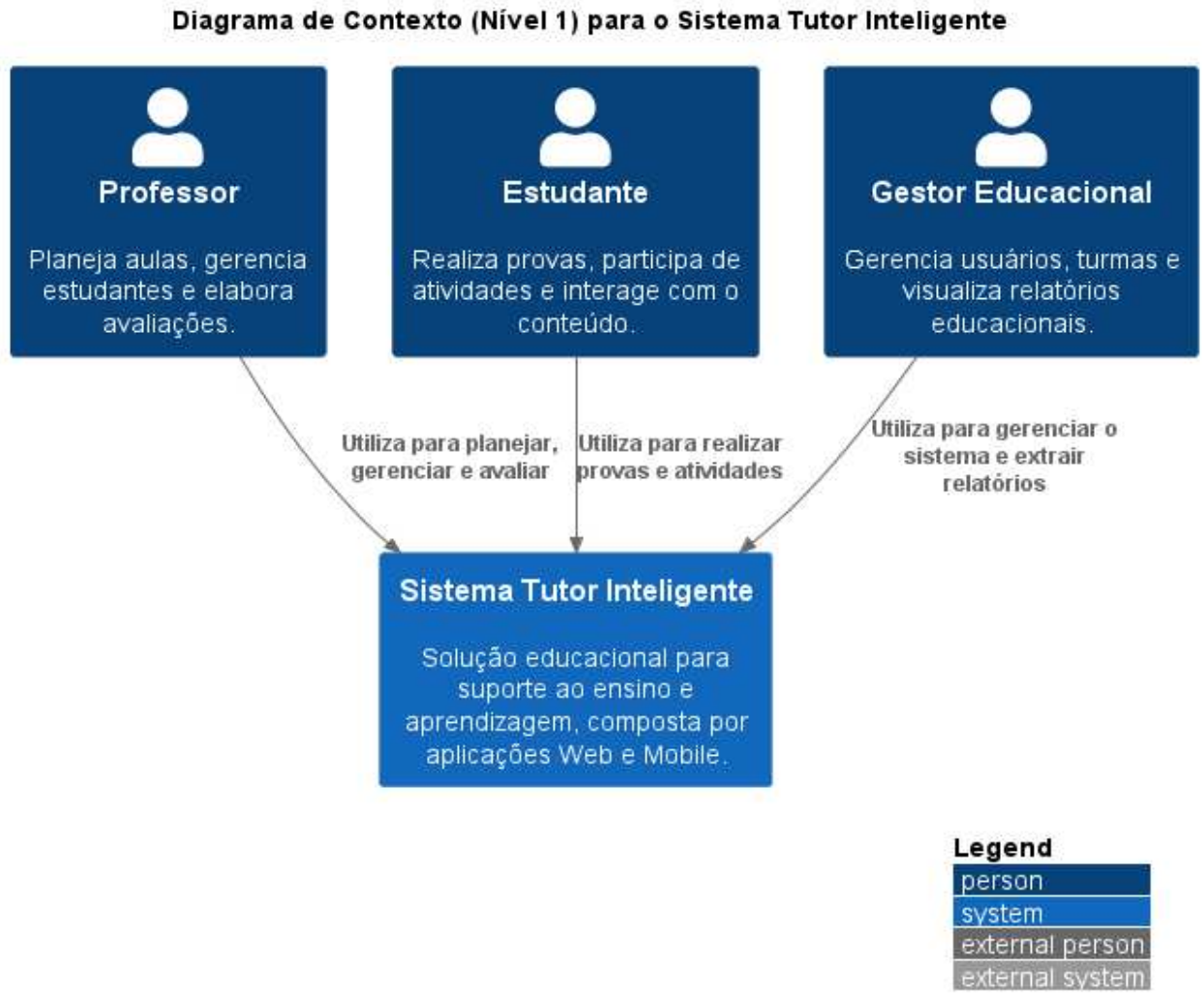
de observabilidade e monitoramento, ressaltando a importância de visibilidade para depuração e auditoria, mas indicando como desafio a complexidade na instrumentação em uma arquitetura *serverless*.

6.2.4 Tarefa 4: Documentação da arquitetura utilizando o modelo C4

Para esse trecho, o LLM foi solicitado a documentar a Arquitetura do STI-Física utilizando o modelo C4, gerando código PlantUML para os diagramas. O processo foi conduzido de maneira iterativa, começando pelo Nível 1 (Contexto) e avançando para os níveis subsequentes conforme instruções do usuário. O LLM demonstrou capacidade interativa ao aguardar comandos antes de expandir os níveis, respeitando fielmente a abordagem progressiva do modelo C4.

No Nível 1, o diagrama de contexto identificou corretamente os três atores principais (Professor, Estudante e Gestor Educacional) e suas interações com o sistema. O código PlantUML foi bem estruturado, utilizando as bibliotecas C4-PlantUML e especificando títulos, legendas e relacionamentos adequados entre os atores e o Sistema Tutor Inteligente.

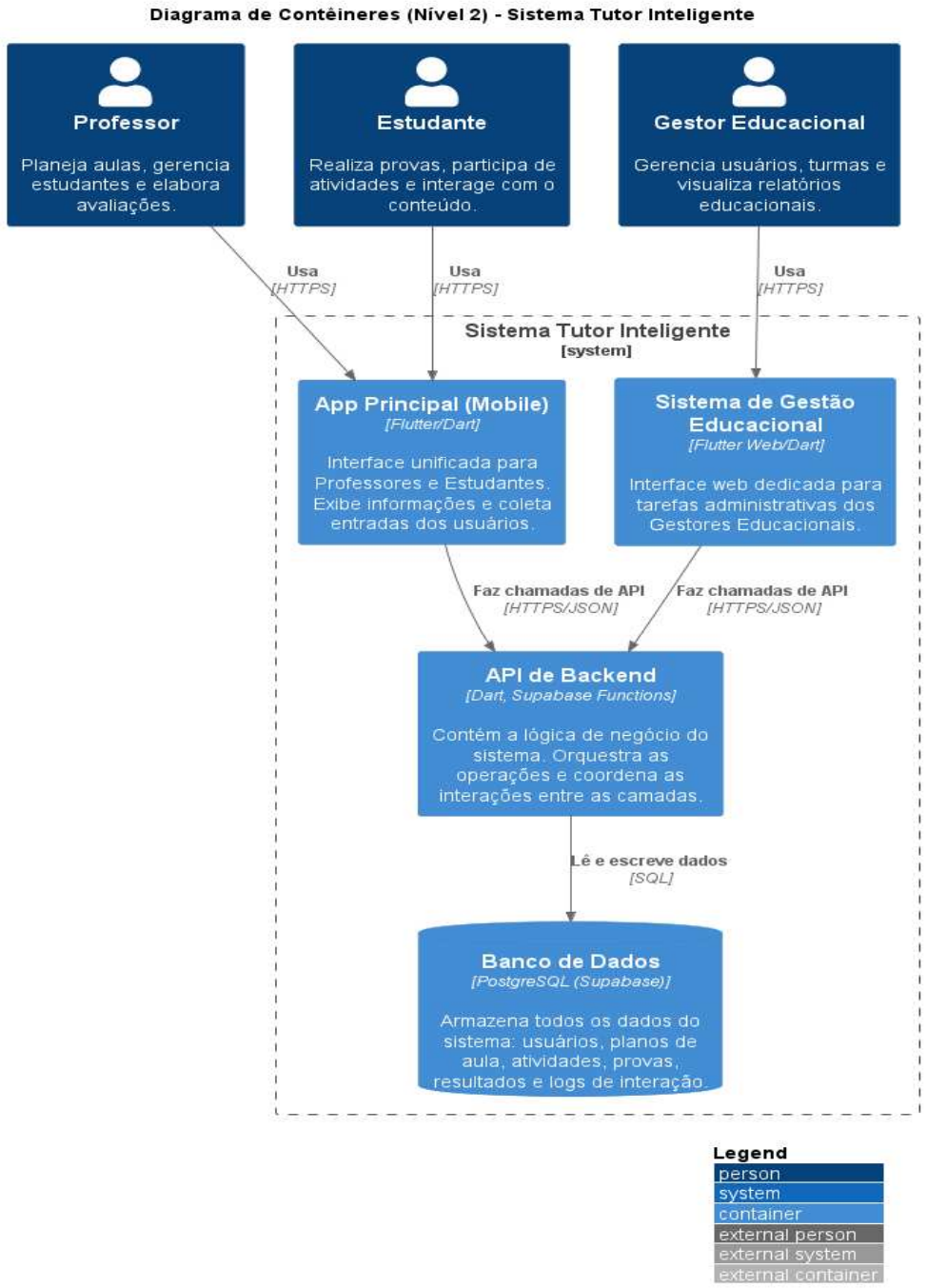
Figura 3 - Diagrama de contexto.



Fonte: Autoria própria (2025).

No Nível 2, o sistema foi dividido em quatro contêineres principais: *App* Principal, Sistema de Gestão Educacional, *API de Backend* e Banco de Dados. O *App* Principal foi desenvolvido em *Flutter/Dart* e serve tanto professores quanto estudantes. A aplicação de gestão é voltada exclusivamente ao gestor educacional. A *API de Backend*, construída com *Dart* e hospedada via funções *serverless* do *Supabase*, centraliza a lógica de negócio, enquanto o banco de dados PostgreSQL é utilizado como repositório único.

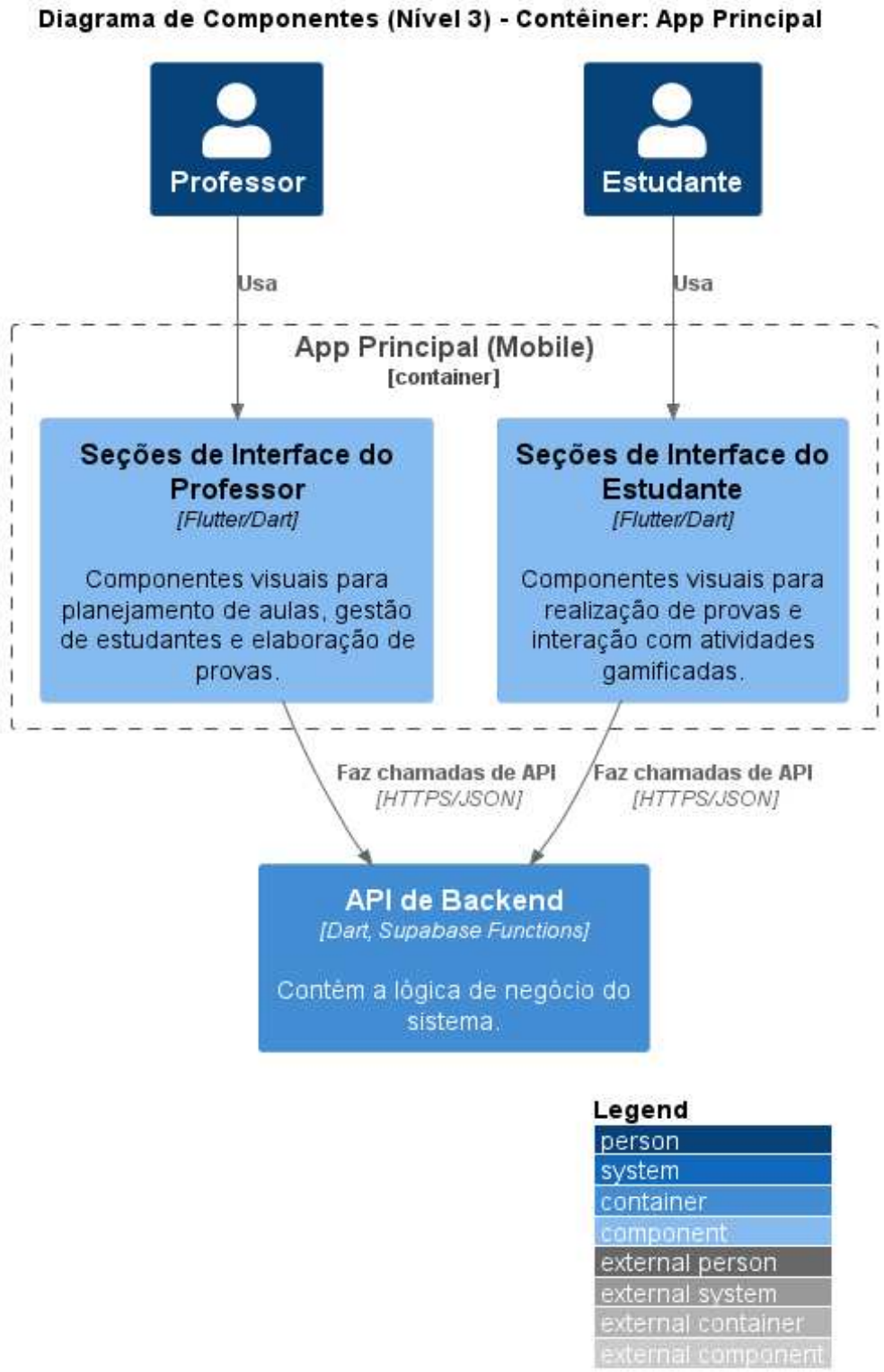
Figura 4 - Diagrama de contêineres.



Fonte: Autoria própria (2025).

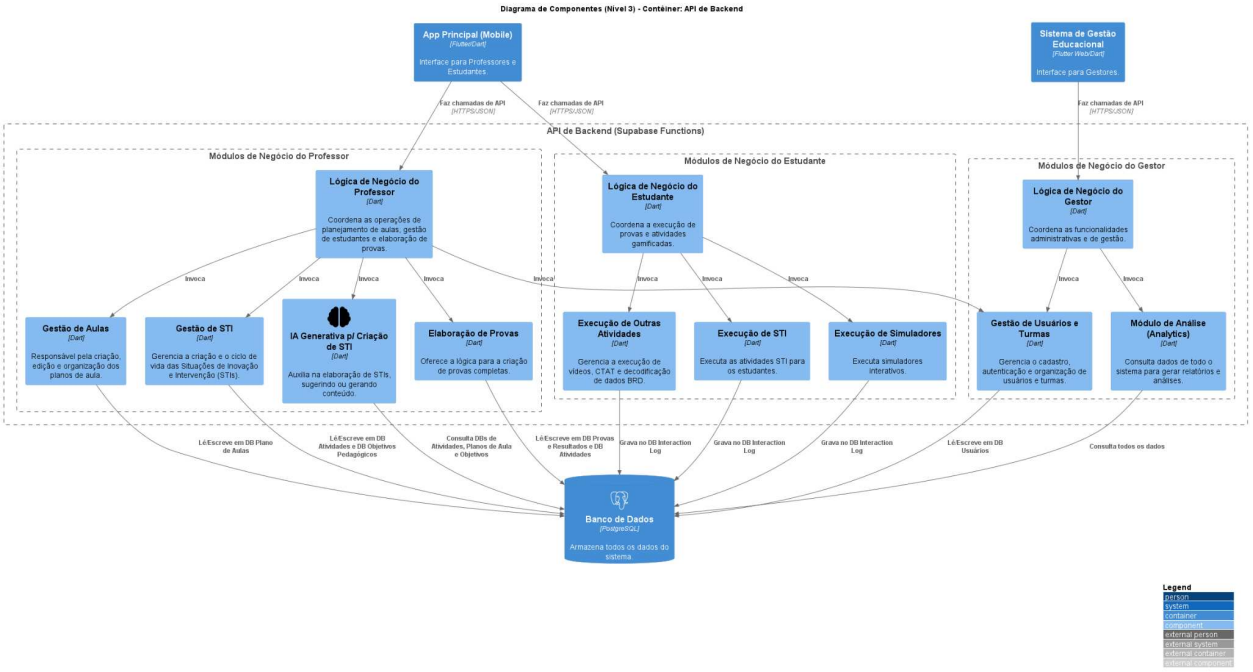
No Nível 3, o LLM detalhou dois contêineres: o *App* Principal e a *API de Backend*. O *App* Principal foi segmentado em seções específicas de interface para professores e estudantes, refletindo a separação de responsabilidades por perfil. A *API de Backend* revelou uma estrutura rica em componentes, organizados em três domínios: professor, estudante e gestor. Os componentes dos respectivos módulos estão organizados em uma arquitetura modular, reforçando a característica original.

Figura 5 - Diagrama de componentes *APP*-Principal.



Fonte: Autoria própria (2025).

Figura 6 - Diagrama de componentes API-Backend.



Fonte: Autoria própria (2025).

É importante ressaltar que o aprofundamento da documentação visual se ateve até o Nível 3 (Componentes). A ausência do Nível 4 (Código) não representa uma lacuna no estudo, mas sim uma decisão metodológica alinhada às melhores práticas propostas por Simon Brown (2022), autor do modelo C4. Ele argumenta que, uma vez que o código-fonte é a fonte definitiva da verdade para este nível de detalhe, criar diagramas que o representem introduz uma sobrecarga de manutenção e uma redundância desnecessária. Portanto, a prática é considerada opcional e, em geral, não recomendada, pois os desenvolvedores podem obter essa informação sob demanda a partir do próprio código.

6.2.5 Tarefa 5: Análise do impacto de mudança na arquitetura

Nesta tarefa, o LLM foi solicitado a analisar o impacto da migração da arquitetura *serverless* baseada em *Supabase Functions* para um *backend* monolítico no Sistema Tutor Inteligente. A avaliação considerou os efeitos sobre a estrutura do sistema, atributos de qualidade e implicações práticas da mudança.

A principal alteração ocorreria nas camadas de Negócio e Componentes, que seriam fundidas em uma aplicação única, eliminando a separação de implantação. A comunicação entre módulos deixaria de ser via rede e passaria a ocorrer como chamadas internas. A camada de

apresentação exigiria refatoração para consumir a *API* agora centralizada, enquanto a camada de persistência teria menor impacto, com centralização no acesso ao banco de dados.

Entre os impactos adicionais, destacam-se a perda de escalabilidade granular, o aumento do risco de falhas devido ao ponto único de falha, e a possibilidade de deterioração da manutenibilidade a longo prazo. Por outro lado, o desempenho poderia ser favorecido pela eliminação da latência de rede, e os custos operacionais poderiam ser reduzidos, dependendo do padrão de uso.

O LLM sugeriu que essa mudança só seja considerada diante de uma justificativa forte como redução de custos ou simplificação operacional.

6.2.6 Tarefa 6: Avaliação de arquitetura

Para essa tarefa, o LLM realizou uma análise técnica da Arquitetura do STI-Fisica, focando em *trade-offs* de atributos de qualidade, impactos adicionais e adequação geral. A arquitetura proposta de quatro camadas utilizando *Flutter/Dart*, *Supabase* foi avaliada. Em desempenho e disponibilidade, o LLM destacou a funcionalidade *offline* como ponto forte, garantindo alta disponibilidade para o aluno. Já o *trade-off* apontado foi o risco de sobrecarga na sincronização automática. Em portabilidade e manutenibilidade, a escolha de *Flutter/Dart* foi elogiada pela portabilidade e a separação em camadas pela manutenibilidade, mas foi identificado uma dependência maior do que esperada com o ecossistema *Supabase*. Em escalabilidade, a arquitetura *serverless* foi vista como ponto forte, mas a centralização do banco de dados no *PostgreSQL* foi apontada como um potencial gargalo. Em segurança, a delegação do *Supabase Auth* foi positiva, porém foi cobrado um detalhamento de implementação e gestão de dados que não foi descrito na arquitetura. Em testabilidade, a modularidade foi um ponto positivo, mas a dependência de serviços externos e a complexidade do *offline-first* foram desafios. Em acoplamento, a modularidade da arquitetura foi elogiada por baixo acoplamento entre os serviços, mas o acoplamento forte ao *Supabase* reforçou a preocupação com a dependência.

Além disso, o LLM apontou impactos adicionais relevantes. Em termos de custos, destacou-se um ganho de produtividade proporcionado pelo uso combinado de Flutter e *Supabase*, mas alertou-se para o risco de aumento de custos à medida que a aplicação escalar, especialmente pelo modelo de cobrança do *Supabase*. Quanto ao cronograma, a Arquitetura acelera o desenvolvimento inicial, porém a complexidade da sincronização *offline* e a dependência de serviços externos podem introduzir desafios em fases posteriores. Os principais riscos identificados foram o forte acoplamento ao *Supabase*, a complexidade do mecanismo *offline-first* e a necessidade de atenção contínua a segurança e a escalabilidade do banco de

dados. Embora essa abordagem arquitetural traga consigo certa complexidade, a modularidade da solução foi vista como um fator que facilita o gerenciamento dessa complexidade. No geral, o LLM considerou a arquitetura proposta adequada aos requisitos do STI-Física.

6.3 Avaliação das propostas

Para avaliar a qualidade das soluções arquiteturais propostas pelo LLM, foi conduzida uma avaliação com quatro membros da equipe do projeto STI-Física que são envolvidos com decisões arquiteturais e desenvolvimento do projeto. Foram apresentadas aos membros duas propostas de Arquitetura de Software para o projeto, a *Proposta A*, baseada na arquitetura original do sistema que foi apresentada na Seção 6.1.2, e a *Proposta B*, gerada pelo Gemini, conforme apresentado nos resultados da Tarefa 1. Os avaliadores não foram informados sobre a origem de cada proposta e foram solicitados a atribuir notas de 1 (Muito Ruim) à 5 (Excelente) para cinco critérios, além de fornecerem um feedback qualitativo sobre as propostas. Os resultados dessa avaliação são apresentados abaixo.

6.3.1 Análise quantitativa

As notas atribuídas pelos quatro avaliadores revelaram percepções distintas para cada proposta. A Tabela 2 abaixo apresenta a média das notas atribuídas pelos quatro membros do projeto para cada critério:

Tabela 2 - Notas da avaliação feita por membros do STI-Física

Critério de Avaliação	Proposta A	Proposta B
Clareza e Compreensibilidade	4.5	4.0
Viabilidade Técnica	4.25	3.0
Facilidade de Manutenção	5.0	3.5
Escalabilidade e Desempenho	4.75	4.0
Inovação da Solução	4.75	4.0

Fonte: Autoria própria (2025).

A Proposta A foi bem avaliada, recebendo notas máximas de dois dos três avaliadores em quase todos os critérios, com destaque para a Facilidade de Manutenção, que recebeu nota máxima de todos. A Proposta B também recebeu boas notas, no entanto, gerou opiniões polarizadas. Enquanto alguns avaliadores a consideraram excelente em Inovação, outros a

avaliaram com nota mínima neste mesmo critério. A maior divergência foi em Viabilidade Técnica e Facilidade de Manutenção, em que as notas variaram drasticamente, sugerindo uma forte incerteza da equipe sobre a complexidade e os riscos de implementação da solução proposta pelo Gemini.

6.3.2 Análise qualitativa

Os comentários qualitativos aprofundaram a análise, explicando as notas atribuídas:

- Percepção sobre a Proposta A: Foi elogiada por sua clareza, boa estrutura em camadas e organização. Os membros da equipe destacaram sua facilidade de entendimento por sua divisão em módulos e a forma de organização da arquitetura. A estrutura em camadas e o uso de tecnologias unificadas foram vistos como pontos positivos que facilitam a manutenção. A única crítica foi a percepção da falta de foco no suporte ao modo *offline*.
- Percepção sobre a Proposta B: O ponto forte mais elogiado foi sua abordagem robusta para o funcionamento offline, que “conversa muito bem com as nossas necessidades”. Contudo, a principal crítica foi a complexidade de implementação. A arquitetura de microsserviços foi vista como uma solução que exigiria mais trabalho e custos para implementação e que traria dificuldades aos desenvolvedores. Houve também uma crítica técnica à escolha do banco de dados, com a sugestão de que outras tecnologias seriam mais adequadas para o ecossistema *Flutter/Dart*, escolhido para o projeto.

A avaliação consolidada dos quatro membros do projeto reforça uma percepção de que a proposta original é mais sólida, objetiva e de fácil manutenção. Em contrapartida, a proposta feita pelo Gemini gerou opiniões divergentes, enquanto reconhecida por alguns como mais inovadora e tecnicamente mais completa para o requisito de funcionalidades *offline*, é também percebida por outros como menos viável e de manutenção complexa.

Este resultado reflete tanto a capacidade dos LLMs de gerar soluções arquiteturais complexas e alinhadas a requisitos específicos, quanto sua dificuldade em ponderar a simplicidade e a viabilidade prática, fatores que são cruciais para a equipe do projeto. A sugestão de um dos avaliadores de que "o ideal seria combinar o melhor das duas: a simplicidade e organização da A com a robustez e resiliência da B" aponta para um dos usos mais promissores da IA na Arquitetura, não como um substituto para o arquiteto, mas como uma ferramenta para gerar alternativas e pontos de vista que, quando filtrados e combinados com a experiência e o contexto da equipe humana, podem levar a uma solução mais equilibrada e robusta.

É importante, contudo, fazer uma ressalva sobre um possível viés nesta avaliação. Como os avaliadores são membros da equipe que já trabalha no desenvolvimento do sistema, sua familiaridade com a arquitetura original, no caso a Proposta A, pode ter induzido uma preferência pela solução já conhecida e compreendida por eles. Este estudo demonstra que a IA pode gerar alternativas valiosas, mas não responde como integrar essa capacidade de forma sistemática e eficaz no ciclo de vida do desenvolvimento de software. Por isso, reforça-se a necessidade de trabalhos futuros que explorem metodologias para essa interação humano-robô, investigando, por exemplo, como diferentes estratégias de engenharia de *prompt* podem guiar os LLMs a produzir soluções que já equilibrem inovação com pragmatismo.

6.4 Análise dos resultados

Esta seção apresenta uma análise SWOT (Forças, Fraquezas, Oportunidades e Ameaças) para cada tarefa executada no estudo de caso, focando na qualidade e capacidade de saída do LLM para cada propósito testado.

6.4.1 Tarefa 1: Proposta de arquitetura

O LLM demonstrou uma excelente compreensão dos requisitos funcionais e não funcionais do STI, especialmente a necessidade de operação *offline* e compatibilidade multiplataforma, que foram pilares da arquitetura proposta. A escolha de Flutter e SQLite para as interfaces e a abordagem *Offline-First* são diretamente alinhadas a esses requisitos críticos. Além disso, a resposta seguiu rigorosamente a estrutura solicitada no *prompt*, apresentando o que foi pedido, o nível de detalhamento para cada componente e a justificativa das escolhas foram notáveis, evidenciando a capacidade do LLM de gerar conteúdo técnico e argumentativo.

A inclusão de alternativas consideradas e a análise de *trade-offs* para cada decisão arquitetural, por exemplo, monólito contra microsserviços foram bem executadas, demonstrando uma capacidade de ponderar sobre diferentes abordagens e seus impactos, o que é crucial para a tomada de decisão de arquitetura. Ademais, a visão de negócio como um todo: custos, tempo, riscos e complexidade, foram bem avaliados com critérios que seriam avaliados em um cenário real, tendo como base os requisitos principais do projeto, mostrando que o LLM pode considerar aspectos além dos aspectos técnicos.

Por outro lado, embora a estrutura geral seja robusta, alguns detalhes da implementação foram explicados rasamente. Por exemplo, a resolução de conflitos no módulo de sincronização *offline* é mencionada, mas sem um mecanismo ou estratégia mais detalhada. Isso pode ser considerada uma limitação inerente ao LLM, que opera em um nível mais conceitual.

6.4.2 Tarefa 2: Comparar duas soluções de arquitetura

A análise da Proposta A sobre a Arquitetura do STI-Física apresenta algumas lacunas e interpretações restritas em relação ao documento original. Embora identifique *trade-offs* importantes, como o acoplamento com o *Supabase* e os desafios do modo *offline*, deixou de considerar aspectos fundamentais da arquitetura descrita como, o funcionamento *offline*, o LLM classifica a proposta A como arquitetura centrada on-line, ignorando elementos presentes na descrição. O documento detalha explicitamente o uso do DB Interaction Log para registrar atividades *offline* e menciona componentes específicos como "Execução de Vídeos *Offline*" e "Execução do CTAT *Offline*", que claramente demandam capacidades de armazenamento e processamento local. Essa abordagem *offline-first* fica evidente na arquitetura, ainda que não sejam detalhados os mecanismos exatos de sincronização.

O LLM fez uma crítica com relação ao desempenho em modo *offline*, sobre não ser possível obter o tempo de resposta de 0.5 segundos, a crítica é pertinente, mas incompleta. A análise não considerou que a escolha do *Flutter/Dart* permite naturalmente a execução de lógica de negócio no lado do cliente, o que poderia atender a esse requisito crítico. Além disso, a possibilidade de pré-carregar conteúdos educacionais no dispositivo não foi levada em conta como estratégia para garantir tempos de resposta adequados mesmo sem conexão. Sobre a manutenibilidade, a ênfase excessiva no acoplamento ao *Supabase*, ofuscou a estrutura modular da arquitetura. A separação em quatro camadas bem definidas e a existência de componentes granulares na Camada 3 demonstram uma preocupação clara com a manutenção e a possibilidade de evolução do sistema. Essa organização permite, por exemplo, que módulos específicos possam ser modificados ou substituídos com impacto limitado no restante do sistema.

Como pontos positivos da análise, destaca-se a clareza na estrutura comparativa entre as propostas e a identificação de *trade-offs* relevantes com relação aos atributos de qualidade esclarecidos na documentação do sistema, mas com adendo aos aspectos e possibilidades que o LLM deixou passar na proposta A.

6.4.3 Tarefa 3: Gerar ADR

Um aspecto importante do experimento foi a capacidade interativa do LLM, que permitiu um fluxo dinâmico na construção de ADRs. O modelo acolheu validações e direcionamentos do usuário, demonstrando habilidade para se ajustar às necessidades. Essa característica mostrou-se essencial para dar suporte ao raciocínio da arquitetura de forma contínua. Além disso, destacam-se a clareza na identificação das decisões arquiteturais, a contextualização precisa e a

explicitação dos impactos de cada escolha. O modelo também mostrou adequação a estrutura imposta pelo usuário, tornando o material padronizado para a documentação. Por outro lado, a seção de conformidade foi pouco explorada, utilizando-se de explicações mais conceituais, o que pode ser um problema futuramente para realizar a manutenção adequada no sistema.

6.4.4 Tarefa 4: Documentação da arquitetura utilizando o modelo C4

O LLM demonstrou uma estrutura metodológica adequada, seguindo corretamente a abordagem iterativa do modelo C4, partindo do contexto e detalhando progressivamente. O código PlantUML gerado estava bem estruturado, utilizando as bibliotecas C4-PlantUML com títulos, legendas e *boundary* apropriados, e a sintaxe estava correta em maioria. Além disso, o LLM identificou precisamente os atores e contêineres essenciais, mostrando uma boa compreensão da arquitetura do sistema. A documentação fornecida foi detalhada para cada nível, listando elementos e relacionamentos, o que facilita a compreensão. No entanto, um ponto crítico observado foi um erro de sintaxe no código do PlantUML (“Container_Boundary” e “ComponentGroup”), em que o LLM deveria ter utilizado apenas “Boundary” para representar o que estaria dentro do módulo, isso comprometeu a geração de diagramas funcionais, necessitando de conhecimentos do autor sobre a linguagem para corrigir o código e gerar os diagramas.

6.4.5 Tarefa 5: Análise do impacto de mudança na arquitetura:

A análise demonstrou uma abordagem equilibrada, cobrindo os impactos técnicos e os aspectos de negócio relacionados à migração arquitetural. A estrutura da resposta foi bem organizada, facilitando a compreensão dos efeitos da mudança em diversos aspectos, como custos, cronograma, riscos e complexidade. Destaca-se a precisão na identificação dos *trade-offs*, com reconhecimento claro de impactos positivos e negativos nos atributos de qualidade.

No entanto, embora o LLM tenha sugerido a necessidade de mitigar impactos negativos, não ficou claro como resolver esses problemas, como o acúmulo de dívida técnica em um monolito de rápido crescimento, que por uma ausência de governança rigorosa pode levar a degradação da arquitetura com o tempo. A avaliação de custos também foi generalista, e teria se beneficiado de cenários comparativos mais especificados.

6.4.6 Tarefa 6: Avaliação de arquitetura

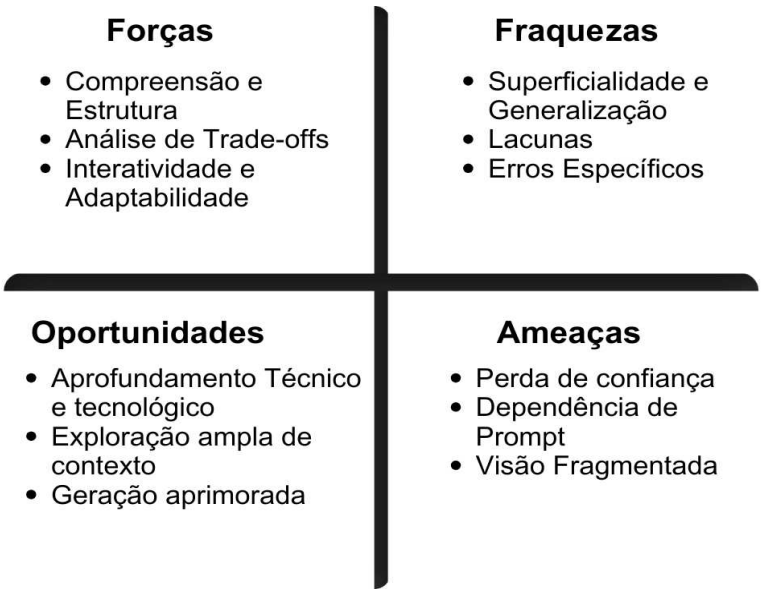
A análise realizada pelo LLM apresentou diversos pontos positivos. Destacou-se por ser abrangente e bem estruturada, abordando com clareza múltiplos atributos de qualidade e impactos adicionais, com identificação consistente e bem articulada dos principais *trade-offs* da arquitetura. Demonstrou ainda uma visão estratégica ao reconhecer de forma precisa o risco de acoplamento ao *Supabase*, indo além de aspectos puramente técnicos. Outro mérito foi o reconhecimento da complexidade inerente à abordagem *offline-first*, especialmente quanto à sincronização e à escalabilidade do banco de dados. Além disso, a avaliação mostrou-se bem alinhada aos requisitos do STI-Física, valorizando corretamente a funcionalidade *offline* como prioridade do sistema.

Por outro lado, alguns pontos poderiam ser aprimorados. Embora os problemas tenham sido bem identificados, as sugestões de mitigação foram genéricas, como o uso de *sharding* para a escalabilidade do banco de dados, que não foi bem explicado. Em alguns trechos, houve repetições de conceitos, comprometendo um pouco a fluidez e a concisão do texto. Além disso, embora a testabilidade tenha sido citada, faltou um nível de detalhamento um pouco maior, sobre como a sincronização *offline* seria testada e como garantir a consistência dos dados em situações de falhas de rede.

6.5 Análise SWOT

Os resultados obtidos nessa análise foram sintetizados em uma matriz SWOT que é apresentada na Figura 7.

Figura 7 - Matriz SWOT



Fonte: Autoria própria

Os pontos destacados na Matriz SWOT são detalhados abaixo:

Forças:

- Compreensão e Estrutura: O modelo demonstrou uma notável habilidade para interpretar os requisitos do sistema e gerar respostas coesas, técnicas e bem-estruturadas, seguindo os formatos solicitados nos *prompts*.
- Análise de *Trade-offs*: Uma força consistente foi a capacidade de realizar análises de *trade-offs*, ponderando diferentes decisões arquiteturais com base em múltiplos atributos de qualidade, como desempenho, escalabilidade e manutenibilidade.
- Interatividade e Adaptabilidade: O LLM se mostrou uma ferramenta altamente interativa, capaz de seguir um fluxo de diálogo, aguardar validações do usuário e adaptar seu processo conforme as instruções, viabilizando um modelo de trabalho colaborativo eficaz.

Fraquezas:

- Superficialidade e Generalização: Uma limitação recorrente foi a tendência à generalização. Em várias tarefas, o LLM apresentou respostas conceitualmente corretas, mas que careciam de profundidade em detalhes de implementação, custos ou estratégias de mitigação.
- Lacunas: Em momentos pontuais, a análise do modelo apresentou lacunas significativas, ignorando ou interpretando de forma restrita informações cruciais presentes nos documentos, o que levou a conclusões incompletas, como discutido na tarefa 2.
- Erros Específicos: Foram observados erros factuais e técnicos, como a geração de código com erros de sintaxe, o que comprometeu a funcionalidade de artefatos e exigiu correção manual, como discutido na tarefa 4.

Oportunidades:

- Aprofundamento Técnico e Tecnológico: Existe uma clara oportunidade de refinar os *prompts* para extrair um nível de detalhe técnico mais profundo, direcionando o LLM para explorar soluções de implementação e tecnologias específicas, superando a generalização.
- Exploração Ampla de Contexto: A ferramenta pode ser utilizada para explorar um leque mais amplo de soluções, incluindo abordagens não convencionais, e para identificar as interconexões entre diferentes decisões arquiteturais, enriquecendo a visão do arquiteto.

- Geração Aprimorada: Há potencial para melhorar a geração de artefatos, não apenas corrigindo erros, mas tornando-os mais completos e acionáveis, como incluir modelos de verificação de conformidade nos ADRs ou detalhes de sintaxe nos diagramas para obter resultados funcionais.

Ameaças:

- Perda de Confiança: A ocorrência de erros, lacunas e generalizações representa uma ameaça direta à confiança do usuário. A imprecisão em análises críticas pode diminuir a credibilidade da ferramenta como um apoio técnico confiável.
- Dependência de *Prompt*: A alta qualidade das respostas está fortemente atrelada à habilidade do usuário em criar *prompts* detalhados e tecnicamente corretos. Essa dependência da engenharia de *prompt* pode ser uma barreira, limitando a eficácia da ferramenta para usuários menos experientes.
- Visão Fragmentada: Se não for bem direcionado, o LLM pode analisar componentes de forma isolada, resultando em uma visão fragmentada da arquitetura que desconsidera o impacto e as interações entre as diferentes partes do sistema.

6.6 Considerações sobre o uso ético da IA na arquitetura de software

Este estudo de caso evidencia que, para além da validação técnica, a integração dos LLMs nas tarefas de Arquitetura de Software exige uma reflexão sobre seu uso ético e responsável. Os resultados demonstraram que a tecnologia pode auxiliar processos, mas suas limitações, como a geração de erros ou a interpretação restrita de contexto, impõem ao arquiteto uma responsabilidade que transcende a supervisão de tarefas.

Nesse sentido, a aplicação de ferramentas de IA deve ser orientada por princípios que garantam a integridade do processo e do produto final. Conforme Sampaio *et al*. (2024), a utilização da IA na pesquisa demanda uma abordagem eticamente orientada, que considere seu potencial impacto nos participantes e na sociedade. Adaptando essa visão para a Arquitetura de Software, o uso de LLMs deve estar alinhado a princípios apontados por Sampaio et al. (2024) como:

- Responsabilidade e Integridade: O arquiteto deve assumir total responsabilidade pelo conteúdo gerado, garantindo que as soluções propostas sejam rigorosas, seguras e honestas, em conformidade com o compromisso com a integridade intelectual.

- Transparência: Todas as atividades que utilizam a IA devem ser transparentes, permitindo a rastreabilidade das decisões e a compreensão de como a ferramenta influenciou o resultado final.
- Soberania Humana: A tecnologia deve ser centrada no ser humano, respeitando a autonomia e a dignidade das pessoas. No contexto do projeto, isso significa que a IA deve servir como uma ferramenta de apoio que aumenta a capacidade do arquiteto, e não como um substituto para seu julgamento crítico e tomada de decisão.

Portanto, conclui-se que o papel do arquiteto, com a proeminência da IA, não é apenas técnico, mas também ético. É sua responsabilidade garantir que essas poderosas ferramentas sejam empregadas de uma maneira que não apenas otimize o projeto, mas que também respeite os princípios fundamentais de uma prática profissional íntegra e responsável.

6.7 Conclusões

Os resultados obtidos neste estudo podem ser considerados promissores. Eles indicam um potencial significativo para que os LLMs, como o Gemini, possam ser ferramentas aliadas do arquiteto de software, seja otimizando tarefas, promovendo insights ou acelerando análises. Os próprios experimentos reforçam de maneira contundente que esses modelos atuam como assistentes, e não como substitutos para o julgamento crítico e a experiência de um arquiteto especialista. Um dos principais fatores a considerar é que nenhum conteúdo deve ser adotado cegamente. A análise revelou desde fraquezas como a superficialidade em detalhes de implementação, que exige a intervenção de um profissional para garantir a profundidade técnica, até erros específicos. Um exemplo claro foi a geração de código para diagramas com erros de sintaxe que os tornaram não funcionais, ou, de forma ainda mais crítica, a interpretação equivocada de uma arquitetura, classificando-a como exclusivamente online ao ignorar possibilidades subentendidas para o funcionamento offline. Um erro de compreensão desta magnitude, se não fosse avaliado, validado e refinado por um arquiteto humano, poderia levar a decisões desastrosas para o projeto. Portanto, a colaboração mais eficiente se dá quando o LLM atua como um catalisador de ideias e o profissional humano mantém o papel de decisor final, responsável pela validação e contextualização.

Vale lembrar que, este trabalho se configura como um estudo inicial, uma primeira visão do que é possível ser alcançado. A confirmação desse estudo e a exploração de novos objetivos exigem a condução de novos experimentos e avaliações mais profundas. Além disso, a escolha e a aplicação desta tecnologia, nesse caso o Gemini, introduz uma variável significativa, de modo que os mesmos *prompts* aplicados a outro LLM pode alterar a natureza dos resultados. Esta limitação aponta diretamente para um caminho claro para trabalhos futuros. Para validar as

diretrizes apresentadas, seria essencial replicar este estudo utilizando outros LLMs, a fim de avaliar sua consistência em diferentes cenários e tecnologias no domínio da Arquitetura de Software.

7. CONSIDERAÇÕES FINAIS

Esse trabalho explorou o potencial e as limitações do uso de Modelos de Linguagem de Grande Escala (LLMs) no apoio às atividades de Arquitetura de Software, um campo tradicionalmente dependente do raciocínio e experiência humana. Diante da crescente complexidade dos sistemas e da necessidade de respostas ágeis no desenvolvimento de software, a pesquisa buscou investigar como os LLMs podem atuar como assistentes inteligentes, otimizando a geração, análise e documentação de soluções arquiteturais. É importante salientar que, este trabalho trata-se de um estudo inicial sobre o tema, devido a complexidade da área e a evolução das tecnologias, faz-se necessário investigar mais profundamente.

O objetivo geral deste estudo foi analisar o uso atual de LLMs em tarefas relacionadas a Arquitetura de Software, identificando seu potencial uso, limitações e estratégias qualificadas. Para alcançar esse objetivo, a metodologia adotada incluiu uma revisão bibliográfica sobre os fundamentos de Arquitetura de Software, aplicações de LLMs na engenharia de software e engenharia de *prompt*. Essa revisão serviu de base para a proposição de diretrizes para o uso de LLMs em tarefas relacionadas com a Arquitetura de Software.

Posteriormente, foram conduzidos experimentos práticos com LLMs, simulando tarefas reais de projeto arquitetural, como proposta e avaliação de arquiteturas, geração de ADRs, documentação visual através do modelo C4, comparação de soluções e análise de impacto de mudanças. A aplicação de diferentes estratégias de engenharia de *prompt* foi um elemento central desses experimentos, visando otimizar a qualidade das respostas. Os resultados obtidos foram analisados e discutidos, buscando identificar a capacidade atual dos LLMs de justificar suas decisões e as limitações observadas no cenário da Arquitetura de Software.

Por conseguinte, uma das principais contribuições deste trabalho é a proposição de diretrizes para o uso de LLMs na Arquitetura de Software. As diretrizes detalhadas no Capítulo 5, foram fundamentadas em referenciais teóricos robustos e nas melhores práticas identificadas nessa pesquisa sobre LLMs. As diretrizes enfatizam a necessidade de estrutura e detalhamento nos *prompts*, a importância da engenharia de *prompt* como pilar fundamental, a promoção da colaboração humano-robô, a orientação para gerar documentações padronizadas e a análise aprofundada de *trade-offs* e atributos de qualidade. Essas diretrizes visam maximizar o potencial dos LLMs como ferramentas de apoio, ao mesmo tempo em que tentam contornar suas limitações já conhecidas.

O estudo de caso realizado com o STI-Física demonstrou a aplicabilidade prática do guia e forneceu *insights* importantes sobre o desempenho dos LLMs em cenários de um projeto de arquitetura real. Através da execução de tarefas descritas no Capítulo 5, foi possível observar que os LLMs tem capacidade de compreender requisitos complexos para gerar propostas

arquiteturais bem estruturadas, incluindo alternativas e análise de *trade-offs*. Em tarefas relacionadas a documentação como a geração de ADRs e a documentação C4, os LLMs foram eficazes ao produzir artefatos consistentes e padronizados, embora com a necessidade de revisão humana para correção de sintaxe ou detalhamento maior, isso evidencia o potencial dos LLMs em reduzir o esforço inicial do arquiteto e oferecer diferentes visões, além dos padrões já consolidados.

Em seguida, a análise SWOT para cada tarefa revelou fraquezas importantes, como a generalização em detalhes de implementação, interpretações restritas de contexto, superficialidade em alguns pontos técnicos e a presença de erros de sintaxe em código gerado. Essas limitações reforçam a necessidade de supervisão humana e pontos a serem trabalhados com *prompts* mais refinados e técnicas de engenharia de *prompt*. A análise também ajudou a identificar ameaças que incluem a alta dependência da engenharia de *prompt*, o risco de soluções genéricas, a geração de artefatos incompletos ou não funcionais, que podem ser associados como uma forma de "alucinação de IA", termo associado a geração de informações incorretas ou equivocadas por um modelo de IA.

Por fim, este trabalho conseguiu mostrar o potencial dos LLMs como ferramentas de apoio valiosas e promissoras no campo da Arquitetura de Software, capazes de automatizar e facilitar diversas tarefas. As diretrizes apresentadas servem como um apoio prático para potencializar a eficácia da colaboração entre profissionais e LLMs. No entanto, os resultados dos experimentos também revelam de forma inequívoca que os LLMs ainda não são autossuficientes. Suas limitações, que vão desde a superficialidade em detalhes técnicos até a geração de artefatos com erros factuais como, código com sintaxe incorreta e interpretação equivocada, reforçam que a intervenção e o julgamento crítico do arquiteto humano são indispensáveis em todas as etapas. Portanto, a colaboração entre humanos e LLMs, onde a inteligência artificial age como um co-agente e o profissional humano mantém a supervisão e a validação final, revela-se como a abordagem mais promissora a curto prazo para alavancar o poder dessas tecnologias.

7.1 Limitações do estudo

É importante reconhecer as limitações inerentes a este estudo, que podem influenciar a generalização e a aplicabilidade dos resultados. Primeiramente, a avaliação dos LLMs foi baseada em um conjunto de tarefas que abordam uma perspectiva geral do problema, isso pode limitar a análise se tratando de cenários mais próximos do nível da implementação. Futuras pesquisas poderiam explorar contextos mais específicos em cenários de arquitetura funcionais.

Em segundo lugar, a análise crítica e a avaliação SWOT das saídas do LLM foram realizadas com base na interpretação e no julgamento do pesquisador. Embora tenha sido

empregado um esforço para manter a objetividade e a fundamentação nas análises, a avaliação da qualidade e da adequação de propostas arquiteturais geradas pelo LLM por profissionais experientes poderiam enriquecer os *insights* oferecidos.

Finalmente, o estudo de caso foi conduzido utilizando a documentação de um projeto em andamento. Embora tenha sido uma abordagem prática para testar as diretrizes, a complexidade e a dinâmica de um projeto consolidado para produção pode apresentar detalhes mais específicos e um embasamento maior na proposta escolhida, o que não foi possível nesse caso.

7.2 Trabalhos futuros

De modo a dar continuidade a este trabalho e a preencher as lacunas e deficiências identificadas em relação ao uso de LLMs na Arquitetura de Software, propomos os seguintes trabalhos futuros:

- Expansão das diretrizes para o uso de LLMs na Arquitetura de Software;
- Realização de novos estudos de caso para avaliar as tarefas propostas;
- Investigação de técnicas mais avançadas para melhorar o desempenho do LLM nas tarefas, como o *fine-tuning*, por exemplo;
- Utilizar o LLM para aplicar métodos de avaliação da arquitetura como o ATAM e o SAAM.

REFERÊNCIAS

AHMAD, Aakash; WASEEM, Muhammad; LIANG, Peng; FAHMIDEH, Mahdi; AKTAR, Mst Shamima; MIKKONEN, Tommi. Towards human-bot collaborative software architecting with ChatGPT. In: **Proceedings of the international conference on evaluation and assessment in software engineering (EASE '23)**, 14–16 jun. 2023, Oulu, Finlândia. ACM, 2023.

BROWN, Simon. **Software architecture for developers: Volume 2 - visualise, document and explore your software architecture**. [S. l.]: Leanpub, 2016.

BROWN, Simon. **The C4 model for visualising software architecture**. [S.l.]: Leanpub, 2022.

CAELEN, Olivier; BLETE, Marie-Alice. **Developing apps with gpt-4 and chatgpt: build intelligent chatbots, content generators, and more**. Sebastopol: O'Reilly Media, 2023.

DHAR, Rudra; VAIDHYANATHAN, Karthik; VARMA, Vasudeva. Can LLMs generate Architectural design decisions? – An exploratory empirical study. In: **2024 IEEE International Conference on Software Architecture (ICSA)**. IEEE, 2024.

EISENREICH, Markus; SPETH, Markus; WAGNER, Stefan. From requirements to architecture: An AI-based journey to semi-automatically generate software architectures. In: **Proceedings of the 2024 IEEE/ACM international conference on software engineering**. 2024. p. 51–60.

GANESH, Sundarakrishnan; SAHLQVIST, Robert. **LLM integration: unveiling architectural patterns and design considerations**. Gothenburg: Chalmers University of Technology, 2024. Master's Thesis in Computer Science.

HARRISON, Neil B.; AVGERIOU, Paris; ZDUN, Uwe. Using patterns to capture architectural decisions. **IEEE Software**, v.24, n. 4, p. 28-34, jul./ago. 2007. Disponível em: https://www.computer.org/software.

JAHIC, Anes; SAMI, Iman. Can large language models support novice designers in making software design decisions? In: **CSE 24–24 Student Group Reports**. 2024. p. 314–316.

OZKAYA, Ipek. Application of large language models to software engineering tasks: opportunities, risks, and implications. **IEEE Software**, v. 40, n. 3, p. 4–8, mai./jun. 2023.

SAMPAIO, R. C.; SABBATINI, M.; LIMONGI, R. **Diretrizes para o uso ético e responsável da inteligência artificial generativa: um guia prático para pesquisadores**. São Paulo: Editora Intercom, 2024.

SOMMERVILLE, Ian. **Engenharia de software**. 10.ª ed. Rio de Janeiro: Pearson, 2019.

VALENTE, Marco Tulio. **Engenharia de software moderna: Princípios e práticas para desenvolvimento de software com produtividade**. [S. l.]: Independente, 2020.

WHITE, Jules et al. A prompt pattern catalog to enhance prompt engineering with ChatGPT. In: **Proceedings of the 2nd international workshop on large language models for software engineering (LLMSE 2023)**. [S. l.]: ACM, 2023. p. 1-8.

WHITE, Jules et al. ChatGPT prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. In: **Proceedings of the 2nd international**

workshop on large language models for software engineering (LLMSE 2023). [S. l.]: ACM, 2023. p. 9-16.