CADERNOS UniFOA 20 ANOS
Edição 56 | 2026
e-ISSN: 1982-1816
Data de submissão: 06/08/2025 Data de Aprovação: 10/09/2026

Modelo de linguagem de grande escala na arquitetura de software: um estudo de caso de um sistema de tutoria inteligente

Large language model in software architecture: a case study of an intelligent tutoring system

[1] Francisco Afonso Matos Pereira Júnior
[2] João Batista de Souza Neto
[1] Reudismam Rolim de Sousa

1 Universidade Federal Rural do Semi-Árido, UFERSA, Pau dos Ferros, RN, Brasil
2 Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, IFRN, Natal, RN, Brasil.

Resumo

A arquitetura de *software* desempenha papel determinante no desenvolvimento de sistemas complexos, orientando escolhas estruturais, padronização de decisões e conformidade com requisitos não funcionais. Recentemente, a integração de Modelos de Linguagem de Grande Escala (LLMs) tem sido explorada para apoiar o processo de concepção arquitetural. Este trabalho apresenta um estudo de caso exploratório que avalia a aplicabilidade do Gemini 2.5 Pro no suporte a decisões e na documentação arquitetural de um Sistema de Tutoria Inteligente para o Ensino de Física (STI-Física). O procedimento metodológico estruturou-se em seis tarefas que abrangem a elaboração de proposta arquitetural, comparação de soluções, geração de *Architectural Decision Records* (ADRs), modelagem visual no *C4 Model*, análise de impacto de mudanças e avaliação de conformidade com atributos de qualidade. Os *prompts* foram elaborados com base em padrões formais e executados repetidamente para verificar a consistência das respostas. Os artefatos gerados foram avaliados por quatro desenvolvedores do projeto em protocolo cego e submetidos a uma análise SWOT. Os resultados indicam que o modelo atua com eficácia como assistente exploratório, demonstrando alta capacidade na estruturação de *trade-offs*, geração de múltiplos ADRs e produção de diagramasPlantUML compiláveis. Contudo, observaram-se limitações técnicas relevantes, incluindo propensão a arquiteturas superdimensionadas (*overengineering*), confusão taxonômica entre banco de dados e ORM, e alucinações de contexto em requisitos de persistência *offline*. Conclui-se que o uso de LLMs proporciona ganhos expressivos de agilidade e padronização, mas exige supervisão humana contínua para assegurar a consistência técnica das soluções.

Palavras-chave:

Arquitetura de *Software*; Modelos de Linguagem de Grande Escala; Engenharia de *Prompt*; Sistema de Tutoria Inteligente; *C4 Model*.

Abstract:

Software architecture plays a decisive role in complex systems development, guiding structural choices, decision standardization, and compliance with non-functional requirements. Recently, the integration of Large Language Models (LLMs) has been explored to assist the architectural design process. This paper presents an exploratory case study evaluating the applicability of Gemini 2.5 Pro in supporting decisions and architectural documentation for an Intelligent Tutoring System in Physics (ITS-Physics). The methodological procedure was structured around six tasks: architectural proposal generation, solution comparison, Architectural Decision Records (ADRs) generation, visual documentation in C4 Model, change impact analysis, and quality attribute compliance assessment. Prompts were designed using formal patterns and executed repeatedly to verify response consistency. Generated artifacts were evaluated by four system developers in a blind protocol and synthesized through a SWOT analysis. Results indicate that the model effectively serves as an exploratory co-agent, demonstrating high competence in structuring trade-offs, generating multiple ADRs, and producing compilable PlantUML diagrams. However, significant technical limitations were observed, including a tendency toward overengineering, taxonomic confusion between database and ORM, and context hallucinations regarding offline persistence requirements. We conclude that LLMs provide substantial agility and documentation standardization, but strictly require human oversight to ensure technical consistency.

Keywords:

Software Architecture; Large Language Models; Prompt Engineering; Intelligent Tutoring System; C4 Model.

1 INTRODUÇÃO

A arquitetura de *software* constitui a espinha dorsal de qualquer sistema computacional de grande porte. Segundo Sommerville (2019), ela afeta diretamente atributos essenciais, como desempenho, robustez, distribuibilidade e manutenibilidade. Decisões arquiteturais inadequadas no início do ciclo de vida costumam acarretar retrabalho custoso e débitos técnicos de difícil mitigação. Ao mesmo tempo, documentar essas decisões por meio de modelos padronizados, visões formais e registros históricos permanece como um desafio crônico na indústria de *software*.

Nesse cenário, os avanços em Inteligência Artificial Generativa e Modelos de Linguagem de Grande Escala (LLMs) vêm transformando diversas etapas da Engenharia de *Software* (Ozkaya, 2023). Modelos recentes demonstram notável proficiência na síntese textual e no raciocínio técnico sobre código e requisitos. No contexto específico da arquitetura de *software*, surgem oportunidades para utilizar tais modelos na exploração de alternativas, na identificação de *trade-offs* e na formalização de artefatos técnicos (Ganesh; Sahlqvist, 2024; Eisenreich *et al.*, 2024).

No entanto, a eficácia do uso de LLMs depende criticamente de estratégias estruturadas de formulação de instruções (engenharia de *prompt*) e da supervisão humana contínua. Para investigar a maturidade dessa colaboração homem-máquina em sistemas com restrições reais de produção, este estudo de caso exploratório avalia o comportamento do modelo Gemini 2.5 Pro no suporte a decisões e na documentação arquitetural do Sistema de Tutoria Inteligente para o Ensino de Física (STI-Física). O Gemini 2.5 Pro foi selecionado por incorporar avanços significativos de raciocínio lógico e suporte a contextos extensos, permitindo processar requisitos complexos e gerar diagramas técnicos.

Com o propósito de orientar e delimitar essa investigação de forma mensurável, formulam-se três Questões de Pesquisa (QP): QP1 (Capacidade e Viabilidade do Modelo): qual é a capacidade e o nível de viabilidade do Gemini 2.5 Pro na execução de seis tarefas arquiteturais fundamentais (proposta de arquitetura, comparação de soluções, geração de ADRs, documentação*C4*, análise de impacto e avaliação de atributos de qualidade)?; QP2 (Correção Técnica e Limitações): quais são as principais limitações, inconsistências e alucinações conceituais manifestadas pelo modelo ao lidar com requisitos não funcionais restritivos e tecnologias de persistência e nuvem?; QP3 (Eficácia dos Padrões de *Prompt*): em que medida a aplicação de padrões consolidados de engenharia de *prompt* contribui para mitigar desvios, estruturar as respostas e manter a completude dos artefatos arquiteturais sob supervisão humana?

2 REFERENCIAL TEÓRICO

2.1 Arquitetura de *Software*

A arquitetura de *software* define a organização fundamental de um sistema, compreendendo seus componentes estruturais, relacionamentos e princípios que orientam seu projeto e evolução (Valente, 2020; Sommerville, 2019). Ela estabelece as bases para o atendimento aos requisitos não funcionais, tais como escalabilidade, manutenibilidade e segurança, atuando como o elo entre os objetivos de negócio e a implementação técnica.

2.2 Decisões Arquiteturais e ADRs

As decisões arquiteturais envolvem a seleção fundamentada de padrões, tecnologias e restrições estruturais que impactam o sistema a longo prazo. O registro formal dessas decisões é essencial para a governança e a rastreabilidade técnica do projeto. Os *Architectural Decision Records* (ADRs) destacam-se

como artefatos leves e padronizados que documentam o contexto, a decisão tomada, suas justificativas e as consequências (*trade-offs*) decorrentes (Harrison; Avgeriou; Zdun, 2007).

2.3 Modelo *C4*

O modelo *C4* (*Context, Containers, Components, Code*) é uma abordagem hierárquica para visualização da arquitetura em diferentes níveis de abstração (Brown, 2016, 2022). Facilita a comunicação técnica ao mapear a solução desde o contexto geral do sistema (Nível 1) até seus contêineres lógicos (Nível 2) e componentes internos (Nível 3), permitindo geração automatizada por meio de ferramentas, como o PlantUML.

2.4 Modelos de Linguagem de Grande Escala

Os Modelos de Linguagem de Grande Escala (LLMs) são redes neurais profundas treinadas em vastos volumes textuais, capazes de processar e gerar linguagem natural e código com alto nível de coerência contextual (Caelen; Blete, 2023). No domínio da Engenharia de *Software*, modelos com capacidades avançadas de raciocínio exibem potencial para atuar como assistentes de síntese, análise exploratória e documentação técnica.

2.5 Engenharia de *Prompt*

A engenharia de *prompt* compreende o conjunto de técnicas para estruturar as instruções submetidas aos LLMs, maximizando a precisão das saídas e minimizando inconsistências (White *et al.*, 2023). O emprego de catálogos de padrões (como definição de personas, restrição de formatos e interações em etapas) é indispensável para obter respostas confiáveis e reprodutíveis em tarefas complexas de engenharia.

3 TRABALHOS RELACIONADOS

A integração de LLMs em tarefas de Engenharia de *Software* tem motivado diversas investigações sobre suas implicações no projeto arquitetural. A engenharia de *prompt* atua como um componente central nesse processo, estruturando as instruções para direcionar a geração de artefatos técnicos. No Quadro 1, sumarizam-se os principais trabalhos relacionados da literatura.

Quadro 1 – Resumo dos trabalhos relacionados da literatura

Referência	Foco Principal	Contribuição Principal	Limitação / Lacuna
Dhar *et al.* (2024)	Geração automatizada de *Architectural Decision Records* (ADRs).	Demonstrou que LLMs aceleram a documentação de decisões e capturam *trade-offs* com instruções claras.	Exige validação de especialistas; suscetibilidade a inconsistências, quando submetido a *prompts* simples.
Ahmad *et al.* (2023)	Uso do ChatGPT como "coarquiteto" em tarefas arquiteturais.	Formalizou a colaboração humano-IA no *design de software*, posicionando o LLM como ferramenta exploratória.	Variabilidade nas respostas e ausência de mecanismos formais para validação de restrições de engenharia.
Eisenreich *et al.* (2024)	Método semiautomático iterativo para derivar arquiteturas a partir de requisitos.	Comprovou a eficácia de abordagens iterativas com *checkpoints* humanos e *prompts* estruturados.	Dificuldades na modelagem de domínio detalhada e na geração precisa de diagramas e códigos técnicos.
Jahić e Sami (2024)	Estado da prática e percepção industrial do uso de LLMs em arquitetura.	Mapeou a adoção industrial: LLMs são vistos como ferramentas inspiradoras, mas com baixa confiabilidade autônoma.	Identificou riscos de decisões que ignoram restrições operacionais e dependências de longo prazo.
Schmid *et al.* (2025)	Revisão Sistemática da Literatura sobre LLMs e Arquitetura de *Software*.	Mapeou 18 estudos na área, categorizando aplicações em geração de arquitetura, ADRs, refatoração e assistentes.	Apontou carência de estudos empíricos multifacetados aplicados a sistemas reais em produção com requisitos restritivos e checagem de conformidade.
Cervantes *et al.* (2026)	Concepção arquitetural assistida por LLM com *Attribute-Driven Design* (ADD).	Integrou o método formal ADD com LLMs e personas de arquiteto, gerando documentação detalhada avaliada por especialistas.	Foco na geração documental e visual estática/comportamental, sem validação empírica de restrições operacionais de sincronização *offline* híbrida ou falhas de *runtime*.
Calamo *et al.* (2026)	Derivação de arquiteturas de microsserviços a partir de histórias de usuário (ArchiLLM).	Automatizou a decomposição de requisitos em microsserviços e mapeamento de padrões (Saga, CQRS) via agentes e RAG.	Foco restrito a sistemas em nuvem em contexto aberto, não contemplando persistência híbrida e requisitos *offline-first* em sistemas reais.
Zhou *et al.* (2025)	Geração e síntese de *design rationale* para decisões arquiteturais com LLMs.	Evidenciou a capacidade de LLMs em sintetizar justificativas técnicas e alternativas de decisão em ADRs.	Avaliação restrita à documentação textual, sem investigar geração de modelos visuais e análise de mudanças operacionais na infraestrutura.

Fonte: Autoria própria (2026).

A literatura evidencia um consenso progressivo: abordagens abertas e sem direcionamento (como *zero-shot* não estruturado) produzem saídas superficiais e inconsistentes, ao passo que estratégias guiadas por padrões de *prompt* e fluxos com supervisão humana contínua entregam resultados expressivamente superiores (Ahmad *et al.*, 2023; Dhar *et al.*, 2024; Eisenreich *et al.*, 2024). Uma revisão sistemática recente mapeou o panorama da área e apontou que o uso de LLMs já se consolida em geração de propostas, síntese de decisões e assistentes técnicos (Schmid *et al.*, 2025), alcançando métodos formais de decomposição de requisitos (Cervantes *et al.*, 2026; Calamo *et al.*, 2026) e captura de justificativas técnicas (Zhou *et al.*, 2025).

Apesar desses avanços, identifica-se uma lacuna evidente na literatura: a grande maioria dos trabalhos prévios restringe-se a tarefas isoladas (exclusivamente geração de ADRs ou apenas decomposição de microsserviços) ou avalia cenários genéricos em nuvem. Poucos estudos submetem um modelo de raciocínio avançado a um fluxo arquitetural completo e integrado (proposta de alto nível, comparação de *trade-offs*, geração de múltiplos ADRs inter-relacionados, modelagem visual hierárquica *C4* em PlantUML, análise de impacto de refatoração e avaliação de conformidade) sobre um sistema real em produção com requisitos

híbridos complexos (como o funcionamento *offline* sincronizado com nuvem). Além disso, há carência de investigações que classifiquem detalhadamente os tipos de erros conceituais e alucinações técnicas do modelo e confrontem as saídas em avaliação cega com a equipe responsável pelo projeto.

O presente trabalho diferencia-se ao conduzir um estudo de caso empírico e aprofundado sobre o STI-Física, aplicando o Gemini 2.5 Pro sob padrões consolidados de engenharia de *prompt* em seis tarefas arquiteturais complementares. O estudo valida a repetibilidade das execuções, realiza uma avaliação cega com quatro desenvolvedores do projeto, expõe e analisa criticamente as limitações e os limites de inferência do modelo (como a interpretação equivocada do suporte *offline* e incompatibilidades de *runtime*) e consolida os achados, por meio de uma matriz SWOT e diretrizes práticas para engenheiros de *software*.

4 METODOLOGIA

Esta pesquisa caracteriza-se como um estudo de caso exploratório de natureza empírica e qualitativa. O procedimento metodológico estrutura-se em quatro etapas: (i) caracterização do ambiente de execução do modelo; (ii) elaboração e aplicação de *prompts* estruturados para seis tarefas arquiteturais; (iii) caracterização do objeto de estudo (STI-Física); e (iv) avaliação cega dos artefatos pela equipe técnica, combinada à análise SWOT.

4.1 Configuração Experimental e Repetibilidade

As execuções foram realizadas com o modelo Gemini 2.5 Pro, por meio da interface *web* oficial do Gemini no modo de *chat*, utilizando as configurações padrão da plataforma. Não foram empregados *system prompts* externos adicionais, assegurando que o direcionamento comportamental derivasse exclusivamente das instruções fornecidas em cada tarefa. Para verificar a estabilidade das respostas frente à variabilidade natural do modelo, cada tarefa foi executada repetidas vezes em novas sessões de *chat* independentes com os mesmos *prompts* e requisitos. Observou-se convergência estrutural consistente entre as execuções, mantendo as decisões técnicas centrais e o padrão de seções exigido. Os artefatos analisados no estudo representam os resultados canônicos obtidos sob esse protocolo.

4.2 Padrões de Engenharia de *Prompt* Utilizados

A formulação dos *prompts* fundamentou-se em quatro padrões catalogados por White *et al.* (2023): (i) *Persona Pattern*: atribuição do papel de arquiteto de *software* sênior para calibrar profundidade e terminologia técnica; (ii) *Template Pattern*:imposição de estrutura de tópicos obrigatórios para garantir a completude documental; (iii) *Output Automater Pattern*:na tarefa de modelagem *C4*, exigência de saída direta em códigoPlantUML processável; e (iv) *Flipped Interaction Pattern*: instrução para o modelo solicitar informações em etapas e aguardar confirmação antes de avançar, assegurando fluxo interativo sob supervisão humana. No Quadro 2, sintetizam-se os objetivos metodológicos e apresentam-se os *prompts* integrais aplicados em cada tarefa.

Quadro 2 – *Prompts* para as diferentes tarefas arquiteturais

Objetivo da Tarefa e Padrões Aplicados	***Prompt* Completo Utilizado**
Tarefa 01: Proposta de Arquitetura Padrões: *Persona*, *Template, Flipped Interaction* Objetivo: Elaboração de uma Arquitetura de *Software* técnica e fundamentada a partir dos requisitos funcionais e não funcionais, avaliando alternativas, *trade-offs* e visão de negócio.	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 vou fornecer a seguir, elabore uma Arquitetura de *Software* técnica e fundamentada para o sistema. A arquitetura proposta deve ser detalhada e estruturada de acordo com os seguintes tópicos: Contexto Geral, Descrição da Arquitetura Proposta, Alternativas Consideradas, Análise Detalhada dos *Trade-offs* por Requisito Não Funcional e Visão de Negócio (custos, prazos, riscos e manutenibilidade). Para garantir esse fluxo de interação, você deve, obrigatoriamente, me pedir autorização para prosseguir antes de qualquer etapa e aguardar o envio dos requisitos antes de iniciar a análise.
Tarefa 02: Comparação de Soluções Padrões: *Persona*, *Template, Flipped Interaction* Objetivo: Comparar criticamente duas propostas de arquitetura frente aos atributos de qualidade e ao contexto do sistema, avaliando *trade-offs* e impactos adicionais.	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.
Tarefa 03: Geração de ADRs Padrões: *Persona, Template, Flipped Interaction*. Objetivo: Criar registros formais de decisões arquiteturais (ADRs) inter-relacionados, seguindo padrão leve (Título, *Status*, Contexto, Decisão, Consequências, Conformidade e Notas).	Você é um arquiteto de *software* experiente, especializado em documentar decisões arquiteturais por meio de 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 essa 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.

Objetivo da Tarefa e Padrões Aplicados	***Prompt* Completo Utilizado**
Tarefa 04: Documentação *C4 Model* Padrões: *Persona, Output Automater, Flipped Interaction*. Objetivo: Gerar código PlantUML progressivo para os diagramas *C4* (Contexto, Contêineres e Componentes) a partir de descrições textuais da arquitetura.	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.
Tarefa 05: Análise de Impacto de Mudanças Padrões: *Persona, Template, Flipped Interaction*. Objetivo: Avaliar e detalhar o impacto técnico, estrutural e operacional de uma alteração arquitetural específica, com recomendações de mitigação.	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.
Tarefa 06: Avaliação de Atributos de Qualidade Padrões: *Persona, Template, Flipped Interaction*. Objetivo: Conduzir avaliação técnica de conformidade arquitetural baseada em atributos de qualidade (desempenho, escalabilidade, disponibilidade, manutenibilidade, segurança, testabilidade, portabilidade e acoplamento).	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.

Fonte: Autoria própria (2026).

4.3 Objeto de Estudo: Sistema de Tutoria Inteligente (STI-Física)

O estudo de caso foi aplicado sobre o Sistema de Tutoria Inteligente para o Ensino de Física (STI-Física), projeto desenvolvido por docentes e pesquisadores da INSTITUIÇÃO (omitida para revisão por pares). O sistema visa à aprendizagem adaptativa em física com o requisito não funcional mandatório de operar em modo *online* e *offline* em locais com conectividade instável.

A arquitetura de referência do STI-Física (designada como Proposta A) organiza-se em quatro camadas lógicas: (i) Apresentação: implementada em Dart com o *framework* Flutter (aplicativos móveis e interface *web*);

(ii) Negócio: funções em nuvem *serverless* no Supabase; (iii) Componentes: módulos de trilhas adaptativas, simuladores e serviços de IA Generativa; e (iv) Persistência Híbrida: a operação *offline-first* é suportada por um banco de dados relacional embarcado SQLite, gerenciado através da biblioteca Drift (camada de persistência/ORM fortemente tipada em Dart), sincronizado com banco central PostgreSQL, gerenciado pelo Supabase (integrando *Auth* e *Storage*).

4.4 Procedimento de Avaliação Humana e Aspectos Éticos

A avaliação envolveu quatro (4) membros técnicos da equipe do STI-Física (com formação em Ciência da Computação/Engenharia de *Software* e atuação no projeto). Em protocolo cego (*blind review*), os avaliadores receberam a Proposta A (arquitetura original em camadas) e a Proposta B (arquitetura em microsserviços gerada pelo Gemini, na Tarefa 1), sem identificação de autoria. Foram atribuídas notas de 1 (Muito Ruim) a 5 (Excelente) em escala *Likert* para cinco critérios fundamentais: Clareza e Compreensibilidade, Viabilidade Técnica, Facilidade de Manutenção, Escalabilidade e Desempenho, e Inovação, complementadas por pareceres qualitativos abertos. **Aspectos Éticos:** A participação ocorreu de forma voluntária, anônima e mediante consentimento livre e esclarecido. Por se tratar de avaliação estritamente técnica de artefatos de *software* produzidos por IA, sem intervenção pedagógica com alunos ou coleta de dados pessoais sensíveis, o estudo atende às diretrizes institucionais de dispensa de submissão a Comitê de Ética em Pesquisa formal.

5 RESULTADOS E DISCUSSÃO

Nesta seção, apresentam-se os resultados obtidos na aplicação dos *prompts* estruturados com o Gemini 2.5 Pro nas seis tarefas arquiteturais, seguidos pela avaliação humana cega, análise SWOT e confronto com a literatura. No Quadro 3, sintetizam-se as saídas geradas, os pontos fortes observados e as limitações técnicas identificadas em cada tarefa.

Quadro 3 – Resumo dos resultados da aplicação do LLM para cada tarefa

Tarefa Arquitetural	Saída Gerada pelo Modelo	Pontos Fortes Observados	Limitações e Alucinações Identificadas
Tarefa 01: Proposta de Arquitetura	Arquitetura de microsserviços com 6 serviços e barramento de eventos (Proposta B).	Estruturação modular completa, separação de responsabilidades e análise abrangente de *trade-offs*.	Proposição de arquitetura com sobrecarga excessiva para o contexto da equipe; sugestão incorreta de *backend* em Dart em funções *serverless* do Supabase (*runtime* Deno/TypeScript).
Tarefa 02: Comparação de Soluções	Análise comparativa entre Proposta A (camadas) e Proposta B (microsserviços).	Identificação precisa de *trade-offs* de manutenibilidade, complexidade operacional e curva de aprendizado.	Alucinação conceitual: o modelo afirmou incorretamente que a Proposta A não possuía suporte *offline*, desconsiderando o SQLite/ Drift informado.
Tarefa 03: Geração de ADRs	Sete (7) registros de decisão arquitetural (ADR 01 a 07) inter-relacionados.	Alto rigor e completude formal nos tópicos de Contexto, Decisão, Consequências e Conformidade.	Dependência de validação humana para calibrar métricas específicas de conformidade operacional.
Tarefa 04: Documentação *C4 Model*	Código PlantUML compilável para Contexto (Nível 1), Contêineres (Nível 2) e Componentes (Nível 3).	Geração sintática correta, permitindo renderização direta dos diagramas visuais hierárquicos.	Confusão taxonômica ao classificar o SQLite como ORM (função exercida pelo Drift) e alta densidade de elementos no Nível 3.
Tarefa 05: Análise de Impacto de Mudanças	Avaliação do impacto da substituição do Supabase por microsserviços customizados em Node. js/PostgreSQL.	Mapeamento detalhado de impactos na camada de autenticação, custos de infraestrutura e esforço de migração.	Tendência a superestimar prazos e custos sem considerar bibliotecas auxiliares de migração.
Tarefa 06: Avaliação de Atributos de Qualidade	Parecer técnico sobre 8 atributos de qualidade (desempenho, escalabilidade, segurança, etc.).	Análise aprofundada de *trade-offs* sistêmicos e recomendações de sustentabilidade arquitetural.	Avaliação predominantemente qualitativa, sem estimativas numéricas de capacidade de carga ou latência.

Fonte: Autoria própria (2026).

5.1 Análise Detalhada dos Artefatos Gerados por Tarefa

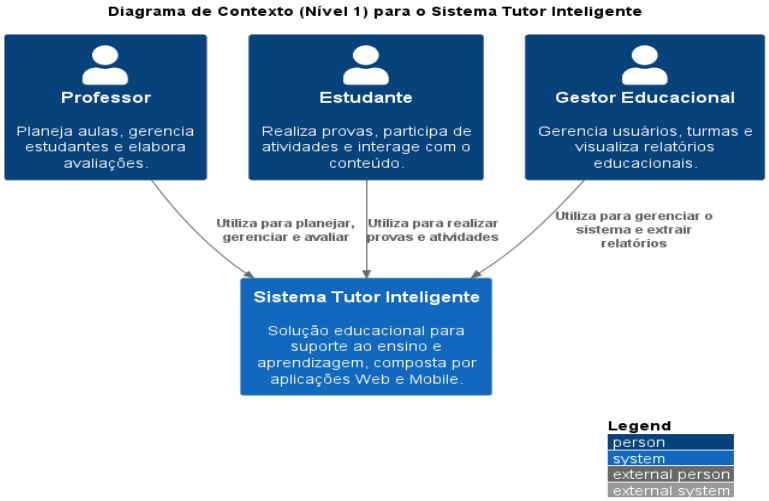
Tarefa 01 (Proposta de Arquitetura): O Gemini 2.5 Pro gerou uma solução detalhada baseada em arquitetura de microsserviços (denominada Proposta B), decompondo o sistema em serviços independentes (Autenticação, Conteúdo/Trilhas, Gamificação, Simuladores, IA Pedagógica e Notificações), comunicando-se de forma assíncrona por mensageria. Embora tecnicamente robusta para grandes escalas, a solução gerou uma sobrecarga de governança desproporcional para o porte da equipe do STI-Física. Adicionalmente, o modelo apresentou uma inconsistência de *runtime* ao sugerir a implementação de funções *serverless* no Supabase diretamente em linguagem Dart, quando o ambiente oficial de execução de *Edge Functions* do Supabase opera nativamente sobre Deno (TypeScript/JavaScript).

Tarefa 02 (Comparação de Soluções): Na comparação entre a Proposta A (arquitetura em camadas original do projeto) e a Proposta B (microsserviços), o modelo identificou com precisão as vantagens da Proposta A em simplicidade, menor custo operacional e facilidade de implantação. Contudo, ocorreu uma alucinação conceitual de contexto: o modelo afirmou que a Proposta A seria estritamente dependente de conectividade à *internet*, ignorando a persistência local com SQLite/Drift fornecida nos requisitos. Esse comportamento evidencia que o modelo pode descartar restrições não funcionais críticas durante etapas comparativas complexas.

Tarefa 03 (Geração de ADRs): Na geração dos registros de decisão, o modelo produziu sete ADRs encadeados com alto nível de completude formal. Destaca-se a formulação do ADR 06 (*Offline-First*), que mapeou adequadamente *trade-offs* de consistência eventual e resolução de conflitos, e do ADR 05 (IA Generativa), que ponderou adaptabilidade pedagógica *versus* custos de inferência e viés algorítmico. A estruturação das consequências e das justificativas de conformidade atendeu às recomendações de Zhou *et al.* (2025) quanto à captura sistemática do *design rationale*, detalhando argumentos de suporte, *trade-offs* e critérios para mitigar violações arquiteturais na implementação.

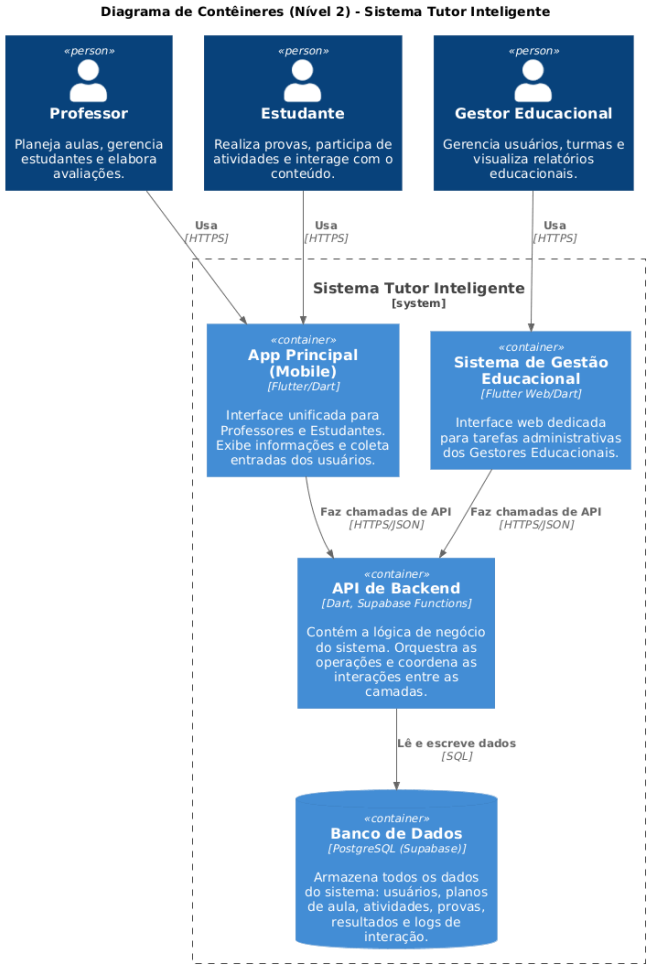
Tarefa 04 (Documentação *C4 Model*): O modelo gerou com sucesso o código PlantUML compilável para o Diagrama de Contexto (Figura 1), Diagrama de Contêineres (Figura 2) e Diagramas de Componentes (Figuras 3 e 4).

Figura 1 – Diagrama de contexto do STI-Física (Nível 1 *C4*)



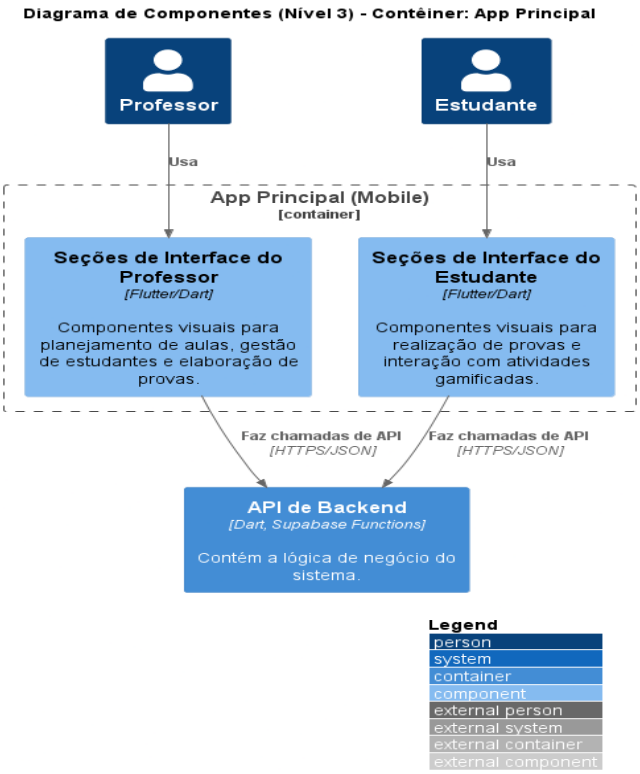
Fonte: *Autoria própria (2026).*

Figura 2 – Diagrama de contêineres do STI-Física (Nível 2 *C4*)



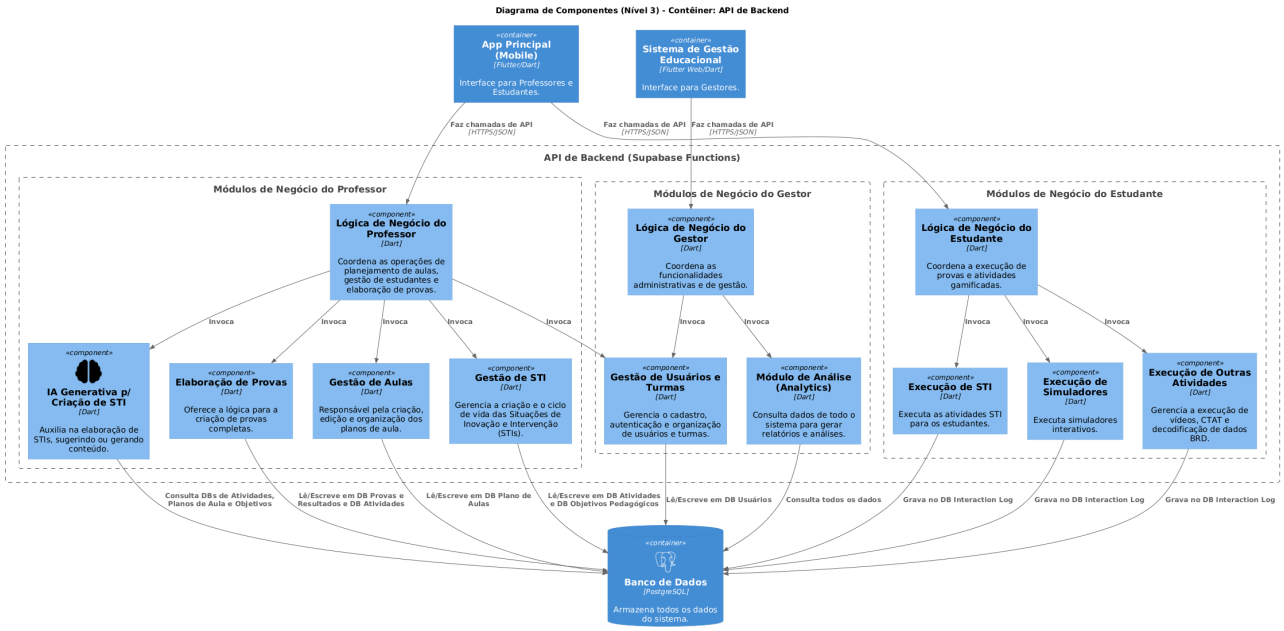
Fonte: *Autoria própria (2026).*

Figura 3 – Diagrama de componentes do aplicativo cliente móvel/*web* (Nível 3 *C4*)



Fonte: *Autoria própria (2026).*

Figura 4 – Diagrama de componentes da API de *Backend* (Nível 3 *C4*)



Fonte: *Autoria própria (2026).*

Os diagramas gerados permitiram visualizar claramente a segregação entre clientes móveis, barramento de eventos e serviços de persistência. Contudo, na descrição dos componentes, o modelo cometeu

um equívoco conceitual de taxonomia ao descrever o SQLite como um "mapeador objeto-relacional (ORM)", quando, na verdade, o SQLite é o motor de banco de dados relacional embarcado, sendo o Drift a biblioteca responsável pelo mapeamento objeto-relacional e persistência tipada em Dart.

Tarefa 05 (Análise de Impacto de Mudanças): Ao analisar a substituição do ecossistema Supabase por uma infraestrutura customizada de microsserviços em Node.js com PostgreSQL autogerenciado, o modelo delineou com precisão os riscos de aumento no custo de desenvolvimento, reescrita das regras de autenticação e perda de sincronização em tempo real nativa, recomendando planos de contingência em fases.

Tarefa 06 (Avaliação de Atributos de Qualidade): A auditoria dos atributos de qualidade mapeou com rigor os impactos em escalabilidade, disponibilidade, segurança e manutenibilidade, ressaltando o acoplamento decorrente do uso de serviços gerenciados de terceiros (*vendor lock-in*) e sugerindo abstrações para mitigar a dependência de fornecedor.

5.2 Avaliação Humana, Análise SWOT e Discussão com a Literatura

A avaliação cega realizada pelos quatro membros técnicos do projeto STI-Física (Quadro 4) demonstrou preferência clara pela Proposta A em Viabilidade Técnica (média 4,75), Facilidade de Manutenção (média 4,75) e Clareza (média 4,50), refletindo a adequação da arquitetura em camadas à realidade da equipe. A Proposta B obteve médias superiores apenas em Escalabilidade/Desempenho (4,50) e Inovação (4,25), sendo penalizada em Viabilidade (2,25) devido à complexidade operacional e aos custos de infraestrutura.

Quadro 4 – Notas da avaliação cega atribuídas pelos membros do STI-Física (Escala de 1 a 5)

Critério de Avaliação	Avaliador 1 (A / B)	Avaliador 2 (A / B)	Avaliador 3 (A / B)	Avaliador 4 (A / B)	Média Proposta A	Média Proposta B
Clareza e Compreensibilidade	5 / 4	4 / 3	5 / 4	4 / 3	4,50	3,50
Viabilidade Técnica	5 / 2	5 / 3	4 / 2	5 / 2	4,75	2,25
Facilidade de Manutenção	5 / 2	4 / 2	5 / 3	5 / 2	4,75	2,25
Escalabilidade e Desempenho	3 / 5	4 / 4	4 / 5	3 / 4	3,50	4,50
Inovação da Solução	3 / 4	3 / 5	4 / 4	3 / 4	3,25	4,25

Fonte: Autoria própria (2026).

Os pareceres qualitativos dos avaliadores reforçaram que a Proposta B, gerada pelo modelo, apresentou sofisticação teórica elevada, mas incompatível com os recursos humanos disponíveis no projeto.

Quadro 5 – Matriz SWOT sobre o uso do Gemini 2.5 Pro na Arquitetura de *Software*

Fatores Positivos	**Fatores Negativos**
Forças (*Strengths*)	Fraquezas (*Weaknesses*)
• Geração rápida de documentação estruturada e abrangente. • Capacidade de sintetizar múltiplos *trade-offs* técnicos por requisito. • Produção direta de código PlantUML compilável e correto. • Estruturação de ADRs encadeados com justificativas sólidas.	• Propensão a alucinações conceituais em requisitos restritivos (*offline*). • Erros taxonômicos (classificar banco de dados como ORM). • Incompatibilidades de *runtime* em tecnologias de nuvem (*serverless*). • Tendência a propor soluções superdimensionadas (*overengineering*).
Oportunidades (*Opportunities*)	Ameaças (*Threats*)
• Redução expressiva no tempo de ideação e exploração arquitetural. • Padronização de artefatos de governança e documentação técnica. • Aceleração da curva de aprendizado de arquitetos juniores via *prompts* estruturados. • Apoio à análise preventiva de impacto de mudanças de escopo.	• Adoção cega de decisões arquiteturais com alucinações em produção. • Aumento de custos por complexidade acidental e sobrecarga de infraestrutura. • Desalinhamento entre a arquitetura gerada e a capacidade operacional da equipe. • Dependência excessiva de modelos estocásticos sem governança humana.

Fonte: Autoria própria (2026).

Discussão Comparativa com a Literatura: Os achados deste estudo corroboram as conclusões de Ahmad *et al.* (2023) e Dhar *et al.* (2024), confirmando que os LLMs operam eficazmente como coagentes de exploração e documentação, mas não substituem o julgamento crítico de um arquiteto humano. Além disso, os resultados alinham-se às observações industriais de Jahić e Sami (2024), que apontam a propensão dos modelos em propor arquiteturas sofisticadas que negligenciam restrições práticas de custo e equipe. O presente estudo complementa as abordagens recentes de Cervantes *et al.* (2026) e Zhou *et al.* (2025), ao demonstrar que, mesmo em modelos de raciocínio como o Gemini 2.5 Pro, a supervisão humana é indispensável para detectar alucinações conceituais de suporte *offline* e incompatibilidades de *runtime serverless*.

6 CONSIDERAÇÕES FINAIS

Este trabalho investigou empiricamente a aplicabilidade do modelo Gemini 2.5 Pro no suporte a decisões e artefatos de Arquitetura de *Software*, utilizando, como objeto de estudo, o Sistema de Tutoria Inteligente para o Ensino de Física (STI-Física). A estruturação metodológica baseada em padrões consolidados de engenharia de *prompt* permitiu responder às três questões de pesquisa formuladas:

Em relação à QP1 (Capacidade e Viabilidade do Modelo), o Gemini 2.5 Pro demonstrou alta aptidão para compreender requisitos de *software* e atuar como coagente exploratório nas seis tarefas investigadas, gerando propostas estruturadas, análises de *trade-offs*, sete ADRs encadeados e código PlantUML compilável para diagramas *C4* em três níveis hierárquicos.

Quanto à QP2 (Correção Técnica e Limitações), a análise crítica e a avaliação humana cega revelaram que, embora o modelo apresente sofisticação teórica elevada, manifesta limitações importantes: propensão a sugerir arquiteturas com sobrecarga operacional excessiva (*overengineering*), inconsistências de *runtime serverless* (como a sugestão de Dart no Supabase), confusão taxonômica entre banco de dados e ORM, e alucinações de contexto (desconsiderando a persistência *offline* local informada nos requisitos). Essas evidências confirmam que o modelo não substitui o julgamento crítico de um arquiteto humano.

No que tange à QP3 (Eficácia dos Padrões de *Prompt*), a combinação dos padrões *Persona*, *Template*, *Output Automater* e *Flipped Interaction* mostrou-se indispensável para guiar o raciocínio do modelo, assegurar a completude estrutural dos documentos e manter o fluxo sob supervisão humana contínua.

Limitações do Estudo: Como limitações metodológicas, destaca-se o foco em um único modelo de linguagem (Gemini 2.5 Pro) e em um único estudo de caso (STI-Física). Ademais, a avaliação humana contou com quatro desenvolvedores membros da própria equipe do projeto, cuja familiaridade prévia com a arquitetura original pode introduzir viés subjetivo, embora mitigado pelo protocolo cego.

Trabalhos Futuros: Para trabalhos futuros, propõe-se: (i) a realização de estudos comparativos com outros modelos de raciocínio avançado (como Claude 3.7 Sonnet, GPT-4o e DeepSeek-R1); (ii) a condução de avaliações com painéis de arquitetos de *software* externos e independentes; (iii) a integração de métodos formais de avaliação arquitetural (como ATAM e SAAM) mediados por LLMs; e (iv) o desenvolvimento de *pipelines* automatizados de checagem de conformidade (*conformance checking*) entre as decisões documentadas e o código-fonte implementado.

Declaração sobre o Uso de Inteligência Artificial: Ferramentas de IA Generativa foram utilizadas exclusivamente para auxílio na revisão textual e formatação do manuscrito. Todos os experimentos, interpretações, análises críticas e conclusões foram concebidos, revisados e validados integralmente pelos autores, que assumem total responsabilidade pelo conteúdo do artigo.

REFERÊNCIAS

AHMAD, A.; WASEEM, M.; LIANG, P.; FAHMIDEH, M.; AKTAR, M.; MIKKONEN, T. Towards Human-Bot Collaborative Software Architecting with ChatGPT. In: INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, 27., 2023, Anais [...] New York: ACM, 2023. p. 279–285. DOI: https://doi.org/10.1145/3593434.3593468.

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

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

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

CALAMO, M.; MONTI, F.; SPAZIANI, F.; LEOTTA, F.; MECELLA, M. From User Stories to Architectures: Using LLM-powered Agents to Design and Improve Microservice-based Software. *IEEE Software*, Washington, v. 43, n. 2, p. 45–54, 2026. DOI: https://doi.org/10.1109/MS.2026.3668669.

CERVANTES, H.; KAZMAN, R.; CAI, Y. Better Together: An LLM-assisted approach to designing software architectures using Attribute-Driven Design. *IEEE Transactions on Software Engineering*, Piscataway, v. 52, n. 3, p. 1–18, 2026. DOI: https://doi.org/10.1109/TSE.2026.3706446.

DHAR, R.; VAIDHYANATHAN, K.; VARMA, V. Can LLMs Generate Architectural Design Decisions? An Exploratory Empirical Study. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE, 21., 2024, Anais [...] Piscataway: IEEE, 2024. p. 79–89. DOI: https://doi.org/10.1109/ICSA59870.2024.00015.

EISENREICH, T.; SPETH, S.; WAGNER, S. From Requirements to Architecture: An AI-Based Journey to Semi-Automatically Generate Software Architectures. In: INTERNATIONAL WORKSHOP ON DESIGNING SOFTWARE, 1., 2024, Lisbon, Portugal. Anais [...] New York: ACM, 2024. p. 52–55. DOI: https://doi.org/10.1145/3643660.3643942.

GANESH, S.; SAHLQVIST, R. *Exploring Patterns in LLM Integration*: a study on architectural considerations and design patterns in LLM dependent applications. 2024. 84 f. Dissertação (Mestrado em Ciência da Computação) – Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden, 2024.

HARRISON, N. B.; AVGERIOU, P.; ZDUN, U. Using Patterns to Capture Architectural Decisions. *IEEE Software*, Washington, v. 24, n. 4, p. 38–45, 2007. DOI: https://doi.org/10.1109/MS.2007.124.

JAHIĆ, J.; SAMI, A. State of Practice: LLMs in Software Engineering and Software Architecture. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE COMPANION, 2024, Anais [...] Piscataway: IEEE, 2024. p. 311–318. DOI: https://doi.org/10.1109/ICSA-C63560.2024.00062.

OZKAYA, I. Application of Large Language Models to Software Engineering Tasks: Opportunities, Risks, and Implications. *IEEE Software*, Washington, v. 40, n. 3, p. 4–8, 2023. DOI: https://doi.org/10.1109/MS.2023.3248401.

SCHMID, L.; HEY, T.; ARMBRUSTER, M.; CORALLO, S.; FUCHß, D.; KEIM, J.; LIU, H.; KOZIOLEK, A. Software Architecture Meets LLMs: A Systematic Literature Review. *arXiv preprint arXiv:2505.16697*, 2025. DOI: https://doi.org/10.48550/arXiv.2505.16697.

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

VALENTE, M. T. *Engenharia de software moderna*: Princípios e práticas para desenvolvimento de software com produtividade. Belo Horizonte: Independente, 2020.

WHITE, J.; FU, Q.; HAYS, S.; SANDBORN, M.; OLEA, C.; GILBERT, H.; ELNASHAR, A.; SPENCER-SMITH, J.; SCHMIDT, D. C. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. In: CONFERENCE ON PATTERN LANGUAGES OF PROGRAMS, 30., 2023, Anais [...] Monticello: The Hillside Group, 2023. DOI: https://doi.org/10.5555/3721041.3721046.

ZHOU, X.; LI, R.; LIANG, P.; ZHANG, B.; SHAHIN, M.; LI, Z.; YANG, C. Using LLMs in Generating Design Rationale for Software Architecture Decisions. *ACM Transactions on Software Engineering and Methodology*, New York, v. 35, n. 2, p. 1–28, 2025. DOI: https://doi.org/10.1145/3785010.