Avaliação do uso de LLMs para Suporte à Tomada de Decisão na Adoção de Padrões de Projeto em Software Orientado a Objetos

Vitor Monteiro Colombo
Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas
Pelotas, Brasil
vmcolombo@inf.ufpel.edu.br

Weslen Schiavon de Souza
Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas
Pelotas, Brasil
wsdsouza@inf.ufpel.edu.br

Lisane Brisolara de Brisolara
Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas
Pelotas, Brasil
lisane@inf.ufpel.edu.br

Brenda Salenave Santana
Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas
Pelotas, Brasil
bssalenave@inf.ufpel.edu.br

Tatiana Aires Tavares
Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas
Pelotas, Brasil
tatiana@inf.ufpel.edu.br

RESUMO

Padrões de projeto são soluções reutilizáveis aplicadas a problemas recorrentes no desenvolvimento de software, visando melhorar a qualidade do projeto. O uso de tais padrões pode trazer uma série de benefícios à arquitetura do software a ser desenvolvido; entretanto, dada a variedade de padrões existentes e seus diferentes propósitos, projetistas inexperientes podem ter dificuldade em identificar padrões apropriados ao contexto. Este trabalho investiga o potencial do uso de grandes modelos de linguagem (LLM, do inglês, *Large Language Model*) na assistência a projetistas no emprego de padrões de projeto em aplicações desenvolvidas sob o paradigma da orientação a objetos. A principal questão discutida e avaliada no artigo é se ferramentas baseadas em LLMs conseguem sugerir e orientar projetistas na escolha e aplicação de padrões em etapas iniciais do projeto. Experimentos foram realizados usando diferentes ferramentas e os resultados foram discutidos e comparados, destacando seus potenciais para apoiar o projeto de arquiteturas de software.

PALAVRAS-CHAVE

inteligência artificial, grandes modelos de linguagem, LLM, engenharia de software, padrões de projeto, arquitetura de software

1 Introdução

Grandes modelos de linguagem (LLMs, do inglês, *Large Language Models*) são exemplos de Inteligência Artificial Generativa, que se destacam na geração de texto, de conteúdo e no processamento de linguagem natural, baseados em aprendizado de máquina [13, 27]. Evoluções recentes nestes modelos têm motivado sua aplicação em diferentes tarefas, abrangendo diversas áreas, como saúde [20], distribuição de energia elétrica [22], desenvolvimento de software [13] e outras tarefas cotidianas [10].

Especificamente na área de Engenharia de Software, estes modelos têm sido usados para geração e refatoração de código, geração de documentação e teste de software, como apontado em [38]. Atualmente, já podem ser encontradas algumas ferramentas que apoiam algumas destas tarefas, propiciando automatização e aumentando a produtividade dos desenvolvedores. A maioria dos esforços desenvolvidos concentra-se em tarefas de codificação. Hoje, a codificação assistida por LLMs já é realidade, sendo suportada por ferramentas como GitHub Copilot [17], Amazon Q Developer [3] e Cursor [5].

No entanto, etapas iniciais do processo de desenvolvimento de software, como a engenharia de requisitos e o projeto de software, também apresentam grande potencial para se beneficiar do uso de Inteligência Artificial. Nesses estágios, a linguagem natural — foco principal dos modelos de linguagem de grande porte (LLMs) — assume um papel central, sendo amplamente utilizada para descrever e especificar requisitos, bem como para registrar decisões de projeto e estruturar informações essenciais do sistema. O suporte de LLMs nessas fases pode contribuir significativamente para melhorar a clareza, consistência e produtividade na documentação e no planejamento de soluções de software.

Nesse contexto, uma das atividades de projeto que mais exige conhecimento especializado é a adoção de padrões de projeto de software, cuja escolha e aplicação adequada impactam diretamente a qualidade e a estrutura do sistema desenvolvido. Os padrões de projeto ganham destaque por fornecerem soluções reutilizáveis para problemas recorrentes no desenvolvimento de software, especialmente no paradigma orientado a objetos. Dentre os diversos tipos de padrões de projeto existentes, destacam-se os padrões GoF (*Gang of Four*) [15], considerados fundamentais por sistematizarem soluções amplamente reconhecidas no desenvolvimento de software orientado a objetos. Esses padrões se mantêm relevantes no contexto atual do desenvolvimento de software, contribuindo para escalabilidade, manutenibilidade e qualidade do software [6].

Estudos recentes [19, 32] sinalizam o potencial de LLMs em tarefas relacionadas ao projeto arquitetural. No entanto, apesar dos benefícios associados à aplicação dos padrões de projeto, os esforços encontrados empregam LLMs na identificação de padrões diretamente no código-fonte (como em Pan et al. [30] e Pandey et al. [31]) ou em diagramas UML [39], enquanto o uso de LLMs para sugestão de padrões em etapas iniciais de projeto permanece pouco explorado. Contudo, mesmo com o potencial dos LLMs para tratar linguagem natural, não foram identificados trabalhos que avaliem de forma sistemática a capacidade de diferentes LLMs em

recomendar padrões de projeto orientado a objetos a partir de informações obtidas pelo levantamento de requisitos.

Diante do exposto, a principal questão de pesquisa que norteia esse estudo é: "Em que medida LLMs podem sugerir padrões de projeto GoF adequados para um sistema orientado a objetos, baseando-se apenas em sua descrição textual inicial?". Para responder a essa questão, as principais contribuições deste trabalho são:

(1) Uma avaliação empírica e comparativa de quatro diferentes LLMs (ChatGPT [29], Gemini [18], DeepSeek [11] e Manus [26]) na tarefa de recomendação de padrões de projeto a partir de descrições textuais de softwares.
(2) A identificação de uma correlação positiva entre o consenso nas sugestões de múltiplas LLMs e a adequação do padrão, validada por especialistas.
(3) A disponibilização dos dados utilizados como base neste trabalho, contendo os prompts, as descrições das aplicações e as respostas geradas por cada ferramenta, que podem ser utilizados em estudos futuros.

Este artigo está estruturado da seguinte forma: a Seção 2 apresenta os fundamentos teóricos sobre os padrões GoF e o uso de LLMs. A Seção 3 discute os trabalhos relacionados e posiciona este estudo na literatura. A Seção 4 descreve a metodologia adotada, enquanto a Seção 5 detalha os experimentos realizados. Em seguida, os resultados obtidos são analisados na Seção 6, e as limitações do estudo são discutidas na Seção 7. Por fim, a Seção 8 apresenta as conclusões e direções para trabalhos futuros.

2 Fundamentação

Padrões de projeto são fundamentais na área de Engenharia de Software, pois oferecem respostas reutilizáveis para desafios recorrentes na criação de sistemas baseados na programação orientada a objetos [7]. Introduzidos primeiramente por Alexander et al. [2], esses padrões favorecem a clareza, a manutenção e a adaptabilidade dos sistemas. A Seção 2.1 revisa o catálogo de padrões conhecido como GoF.

Na área de Inteligência Artificial, os Grandes Modelos de Linguagem representam uma sub-área importante, que desenvolveu-se recentemente. Estes modelos mostraram-se capazes de compreender e criar texto com grande precisão [37] e vêm sendo aplicados em diferentes áreas. A Subseção 2.2 revisa alguns conceitos sobre estes modelos e detalha as ferramentas que serão empregadas neste trabalho.

2.1 Padrões GoF

Os padrões de projeto oferecem respostas reutilizáveis para desafios comuns enfrentados no desenvolvimento de software orientado a objetos. De acordo com Gamma et al. [15], tais padrões apresentam estruturas que ilustram a interação e a organização entre classes e objetos [16].

O catálogo GoF, criado por Gamma [15], é amplamente considerado a principal fonte de referência para padrões de projeto orientados a objetos. Este catálogo organiza vinte e três padrões de projeto em três grupos: criacionais, estruturais e comportamentais, visando facilitar a reutilização, a flexibilidade e a manutenibilidade de sistemas de software. A Tabela 1 apresenta os padrões de projeto, agrupados pelas categorias indicadas no catálogo GoF.

Tabela 1: Padrões de Projeto segundo o Catálogo GoF [15].

Categoria	Nomenclatura
Criacionais	Singleton, Factory Method, Abstract Factory, Builder, Prototype
Estruturais	Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy
Comportamentais	Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, *Strategy* , Template Method, Visitor

Nessa seção será apresentada uma revisão dos padrões que serão discutidos neste trabalho, a saber: *Strategy* , *Observer*, *Composite*, *Factory Method*, *Proxy*, *State* e *Builder*. Esses são exemplos clássicos de padrões GoF que têm sido muito empregados em soluções de software para lidar com problemas recorrentes de forma eficaz e estruturada.

Dentre os padrões criacionais, este estudo destaca o padrão *Factory Method* e o padrão *Builder*, cujos empregos serão discutidos nos experimentos realizados. O padrão *Factory Method* estabelece uma interface destinada à criação de instâncias de objetos, transferindo para as subclasses a tarefa de decidir qual classe concreta deve ser instanciada. Para o código que usa o método fábrica não haverá diferença entre os objetos retornados por diferentes subclasses. Essa abordagem é aconselhada quando há menção a mudanças na criação de objetos ou à necessidade de expansão [15]. O padrão *Builder* isola a lógica de construção da representação de objetos complexos, permitindo que o mesmo processo de criação produza diferentes configurações de objetos. Segundo Gamma [15], sua aplicação é recomendada em cenários onde os requisitos indicam múltiplas variações estruturais ou etapas específicas na montagem do objeto.

Dos padrões estruturais, este estudo revisa os padrões *Composite* e *Proxy*. O *Composite* organiza elementos em forma de árvore para ilustrar hierarquias de composição parte-todo. O emprego deste padrão possibilita unificar o tratamento tanto de objetos isolados quanto de objetos compostos de maneira consistente, sendo valioso em situações que demandam a representação de estruturas hierárquicas [15]. Já o padrão *Proxy* oferece um substituto para um objeto original, geralmente que oferece um serviço. O proxy controla o acesso ao objeto original, permitindo que uma lógica seja executada antes que a solicitação chegue ao objeto original, gerenciando o acesso a este. O padrão costuma ser utilizado em situações que requerem controle de acesso, adiamento de ações ou otimização de recursos [15].

Neste estudo, quatro padrões comportamentais serão discutidos; são eles: *Command*, *Observer*, *Strategy* e *State*. O padrão comportamental *Command* define um objeto que representa uma requisição, organizando os parâmetros necessários para sua execução, permitindo passar uma requisição como argumento em uma chamada. Este padrão permite organizar pilhas de comandos ou ainda tratar requisições de desfazer ou refazer.

O padrão comportamental *Strategy* possibilita a definição de uma família de algoritmos. Um mesmo objeto pode estar relacionado a diferentes algoritmos, permitindo alterar o seu comportamento em tempo de execução. Este padrão é empregado quando há necessidade de intercambiar o algoritmo empregado, permitindo comportamentos diferentes conforme condições externas [15].

O padrão comportamental *Observer* permite implementar mecanismos de notificação para múltiplos objetos quando eventos acontecem em um determinando objeto. Este padrão define uma relação de um-para-muitos entre entidades, de forma que, ao alterar o estado de um objeto, todos os seus dependentes sejam informados automaticamente, sem necessidade de intervenção manual. Tal característica é relevante para situações que exigem atualizações em tempo real ou a gestão de eventos [15].

O padrão *State* permite que um objeto altere seu comportamento em resposta a alterações em seu estado interno, fazendo com que pareça que sua classe foi alterada. Essa abordagem é valiosa em situações que demandam variações de comportamento vinculadas a estados diferentes [15].

O padrão *Command* encapsula uma solicitação como um objeto, permitindo parametrizar clientes com diferentes solicitações, enfileirar ou registrar operações e suportar desfazer operações. É especialmente aplicável em requisitos que descrevem ações desacopladas do objeto que as executa, como pedidos de usuário ou operações reversíveis [15].

2.2 Grandes Modelos de Linguagem

Grandes Modelos de Linguagem, como os desenvolvidos por empresas como OpenAI, Google e outros centros de pesquisa, são sistemas de aprendizado profundo que utilizam vastas quantidades de dados textuais para seu treinamento [1]. Baseiam-se, em sua maioria, na arquitetura Transformer [35], permitindo que possam gerar, resumir, classificar e analisar textos em linguagem natural.

Grandes Modelos de Linguagem mostram habilidade em interpretar textos complexos, deduzir conexões semânticas e executar tarefas de categorização, o que os torna valiosos para a identificação de padrões de design em requisitos. Pesquisas recentes [34] indicam que modelos de linguagem treinados com diretrizes adequadas podem ser empregados para identificar padrões estruturais, semânticos e funcionais em descrições feitas em linguagem natural. Todo LLM opera a partir de prompts — instruções textuais que orientam sua geração de respostas. O conteúdo e a clareza desses prompts influenciam diretamente a qualidade e a precisão das respostas geradas, sendo que variações na formulação, no nível de detalhamento e no uso de pistas linguísticas específicas podem alterar significativamente os resultados [24].

A aplicabilidade dos LLMs na sugestão de padrões de projeto deriva de sua capacidade de reconhecer padrões contextuais em descrições textuais. Treinados com vastos corpora de texto e código-fonte, esses modelos aprendem a associar problemas recorrentes, descritos em linguagem natural, a soluções de design estabelecidas que aparecem em documentações, tutoriais e bases de código. Portanto, neste trabalho parte-se da hipótese que um LLM pode atuar como um "assistente de projetista"ao identificar pistas em uma descrição de software e mapeá-las para padrões de projeto GoF relevantes, como, por exemplo, o *Observer* para sistemas reativos ou o *Composite* para estruturas hierárquicas. As ferramentas a seguir foram selecionadas por representarem diferentes arquiteturas e bases de conhecimento para avaliar essa capacidade. Selecionamos dentre os principais, quatro para abordar neste artigo: (1) ChatGPT; (2) Gemini; (3) DeepSeek; e (4) Manus, descritos na sequência.

(1) **ChatGPT (OpenAI)**: Criado pela OpenAI, este é um dos Grandes Modelos de Linguagem mais reconhecidos e utilizados. Fundamentado na linha de modelos GPT (Transformador Generativo Pré-treinado), ele é capaz de executar atividades de raciocínio, interpretar significados e produzir textos de maneira coesa. Sua aplicação tem sido testada em várias áreas da engenharia de software. Apesar de apresentar vantagens como escalabilidade e eficiência, ainda lida com obstáculos relacionados a questões éticas [23].
(2) **Gemini (Google)**: Introduzido pela Google DeepMind em dezembro de 2023, é um sistema de inteligência artificial multimodal capaz de lidar com texto, imagens, áudio e vídeo. Ele é oferecido em três variantes — Nano (ideal para dispositivos compactos), Pro (balanceado) e Ultra (o mais potente) — podendo superar outros modelos, incluindo o ChatGPT-4, em 30 dos 32 testes do benchmark MMLU [25].
(3) **DeepSeek**: É um projeto de inteligência artificial dedicado à criação de LLMs de código aberto, priorizando eficiência e raciocínio sofisticado. Entre seus principais modelos estão o DeepSeek-LLM, V2, V3 e R1, que se destacam em matemática, programação e tarefas complexas. O modelo DeepSeek-V3, que possui 671 bilhões de parâmetros e foi treinado com 14,8 trilhões de tokens, emprega a arquitetura Mixture-of-Experts (MoE) e Multi-Head Latent Attention (MLA), superando outros modelos de código aberto e competindo com o GPT-4o em benchmarks como MMLU e MATH-500 [28].
(4) **Manus**: Introduzido em março de 2025 pela empresa chinesa Monica.im, é um sistema autônomo de inteligência artificial criado para realizar tarefas complexas com pouca intervenção humana. Ele utiliza modelos avançados, incluindo Claude 3.5 Sonnet[4] e Qwen [8], para processar diferentes tipos de dados (texto, imagens, código) e se conecta a ferramentas externas, como navegadores e bancos de dados [33].

3 Trabalhos Relacionados

Aprendizado de máquina, mais especificamente os LLMs, têm sido explorados em várias aplicações, inclusive em Engenharia de Software. Em Zhang et al. [39], os autores propuseram um framework para reconhecer padrões em diagramas UML. Para tal, foi treinada uma Rede Neural Convolucional (CNN, do inglês, *Convolutional Neural Network*) que atua como classificador. O treinamento utilizou um conjunto de imagens de diagramas de classe UML, obtidas da internet, representando cinco padrões de projeto específicos: *Adapter*, *Bridge*, *Command*, *Iterator* e *Strategy* . Com o classificador treinado, é possível submeter diagramas UML ao modelo e receber o nome do padrão correspondente. A abordagem proposta pelos autores foi mais efetiva para padrões estruturais, mas tem dificuldades com padrões comportamentais, uma vez que se baseia unicamente nos diagramas de classes que representam uma visão estrutural das classes.

Diferente da abordagem de Zhang et al. [39], estudos recentes têm explorado o uso de LLMs diretamente sobre código-fonte. Em Pandey et al. [31], os autores avaliaram a capacidade de LLMs em identificar padrões de projeto em códigos Java. Os resultados são promissores, especialmente considerando que há menos esforço para identificar regras específicas de padrões e não há pré-treinamento dos modelos (isto é, os modelos não foram ajustados especificamente para identificação de padrões de projeto). Entretanto, padrões de projeto implementados em outras linguagens (eg., C++, Python) podem ser diferentes em estrutura e comportamento, o que limita o escopo dos experimentos do estudo.

Em Pan et al. [30], os autores investigaram a capacidade de diferentes LLMs em reconhecer padrões de projeto em códigos implementados em Python e Java. O conjunto de dados utilizado continha implementações de doze padrões distintos, organizados em três níveis de complexidade: simples, moderado e difícil. Os resultados indicaram que o aumento da complexidade dos códigos e dos próprios padrões comprometeu significativamente o desempenho dos modelos. Além disso, observou-se uma acurácia inferior nas análises de código em Python quando comparadas às de Java — um resultado possivelmente atribuído a características sintáticas específicas da linguagem Python ou à menor representatividade de exemplos envolvendo padrões de projeto nessa linguagem nos dados de treinamento utilizados pelos modelos. Os resultados desse estudo corroboram e ampliam as observações de Pandey et al. [31], indicando que os LLMs apresentam dificuldades na identificação de padrões de projeto mais abstratos e conceitualmente semelhantes — por exemplo, o padrão *Singleton* foi frequentemente confundido com *Facade, Proxy e Command*. Além disso, os modelos obtiveram desempenho superior ao analisar códigos em Java, em comparação com implementações em Python.

Apesar das limitações observadas na identificação de padrões de projeto, o ChatGPT tem sido amplamente adotado por desenvolvedores em diversas tarefas ao longo do processo de desenvolvimento de software [9]. Dentre essas tarefas, destacam-se a implementação de novas funcionalidades e o gerenciamento da qualidade do código — atividades que podem se beneficiar diretamente da aplicação adequada de padrões de projeto. Em [14] os autores avaliaram o emprego do ChatGPT para apoiar na tomada de decisão e definição de funções objetivo no projeto arquitetural de Linhas de Produtos de Software.

Além disso, Jahić and Sami [19] investigou o uso do ChatGPT como apoio em atividades de arquitetura de software, por meio de um survey com profissionais da área. Os participantes relataram que o modelo sugeriu padrões de projeto que eles ainda não haviam considerado, evidenciando seu potencial como ferramenta de apoio ao raciocínio arquitetural e ao aumento da produtividade. Esse cenário reforça a relevância de investigar como LLMs, para além do ChatGPT, podem ou não auxiliar a incorporação desses padrões em contextos de desenvolvimento orientado a boas práticas.

A literatura existente demonstra um interesse crescente no uso de IA para tarefas de Engenharia de Software, com um foco notável na identificação de padrões de projeto a partir de artefatos de estágios tardios do processo de desenvolvimento, como código-fonte ou diagramas UML. Entretanto, a revisão realizada não encontrou estudos que avaliassem e comparassem a eficácia de diferentes LLMs no suporte à aplicação de padrões de projeto em etapas iniciais de projeto onde apenas descrições textuais em linguagem natural estão disponíveis.

Este trabalho endereça diretamente essa lacuna ao fornecer a primeira avaliação sistemática e comparativa de diferentes LLMs na tarefa de sugerir padrões de projeto com base em descrições textuais de aplicações, validando as recomendações através de análise de especialistas e investigando o consenso entre as ferramentas como um indicador de qualidade.

4 Metodologia

Esta seção apresenta a metodologia empregada nos experimentos. O primeiro passo consistiu na definição dos prompts a serem utilizados, seguido da elaboração das descrições dos projetos de software — um para cada experimento, sendo três ao todo. Em seguida, os experimentos foram realizados utilizando o mesmo conjunto de prompts em diferentes ferramentas baseadas em LLMs. Os resultados gerados por cada ferramenta, para cada experimento, foram então coletados e avaliados por especialistas. A Figura 1 ilustra e sintetiza a metodologia empregada nos experimentos.

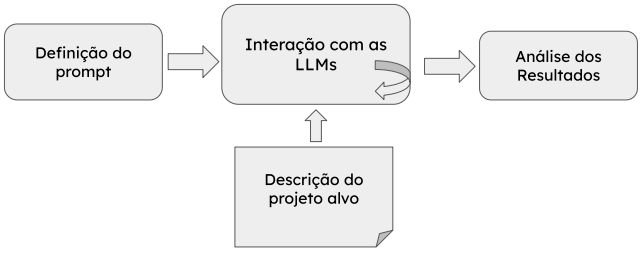

Figura 1: Metodologia usada nos experimentos

Considere que sou um projetista de software atuando no projeto **"título do projeto"**

Gostaria de auxílio para identificar e explorar o emprego de padrões de projeto orientado a objeto (design patterns)

Gostaria de sugestões de padrões até três padrões de projeto que melhor atendam às características, funcionalidades e objetivos mencionados na descrição do software.

Para cada padrão sugerido, apresente
- Padrão: [nome]
- Categoria: [Criacional / Estrutural / Comportamental]
- Aplicação no projeto: [...]
- Benefícios: [...]

Considerando que padrões também podem trazer complexidade ao código, indique apenas os padrões mais benéficos no contexto.

"Descrição textual detalhada:"

Figura 2: Prompt usados nos experimentos

Com base em alguns experimentos iniciais, foi otimizado o prompt a ser empregado. Na definição do prompt, também considerou-se orientações apresentadas em White et al. [36] para exploração de padrões arquiteturais. O prompt final incluiu as seguintes características:

- Limitação no número de padrões a serem sugeridos pelos modelos ("até **três** padrões de projeto").

- Inclusão de informações contextuais ("Considere que sou um **projetista de software** atuando no projeto 'título do projeto'").
- Definição do formato esperado da resposta ("Para cada padrão sugerido, apresente **Padrão** [nome], **Categoria** [Criacional / Estrutural / Comportamental] , **Aplicação no projeto** [...] e **Benefícios** [...]").
- Objetivos de qualidade do software ("Meu objetivo ao empregar padrões de projeto é melhorar a **manutenibilidade** para facilitar futuras modificações do software e obter **escalabilidade**").

O objetivo da aplicação de características como a limitação do tamanho, do formato e do escopo das respostas foi orientar as ferramentas utilizadas a produzirem saídas mais alinhadas aos requisitos. Além disso, a análise de um número elevado de padrões possíveis seria inviável no contexto de um projeto real. Com tais limitações, buscamos focar então na inclusão de informações contextuais relevantes que, aliadas à explicitação dos objetivos de qualidade, podem contribuir para respostas mais precisas e relevantes em relação a conceitos abstratos [12].

A Figura 2 ilustra a estrutura de *prompt* empregada. Onde lê-se "título do projeto", e "descrição textual detalhada"serão substituídos por textos construídos para cada experimento. O primeiro estudo analisa a proposta de um projeto de uma agenda para gerenciamento de eventos. O segundo uma proposta de uma ferramenta de CAD e o terceiro contexto analisado é uma ferramenta de gestão de projetos. Os textos completos utilizados estão disponibilizados no link citado nos recursos online.

Como LLMs foram empregados quatro modelos distintos, sem treinamento especializado (*fine-tuning*) e sem nenhuma modificação em seus parâmetros. A primeira sugestão feita pela ferramenta foi salva para análise e comparação. Na repetição do processo, para o experimento seguinte, foi solicitado que a ferramenta desconsiderasse o contexto do projeto anterior no tratamento da requisição corrente. A Tabela 2 apresenta detalhes das ferramentas empregadas, destacando a versão do modelo empregado. Versões pagas do Gemini Pro e do ChatGPT foram usadas e versões gratuitas do Manus [1] e do Deepseek foram avaliadas.

Tabela 2: Detalhamento das ferramentas empregadas

Ferramenta	Versão
ChatGPT	GPT-4-turbo - versão paga
Gemini	2.5 pro - versão paga
Manus	versão gratuita
DeepSeek	V3 - versão gratuita

5 Experimentos

Para avaliar o uso de LLMs como ferramentas de apoio à decisão no emprego de padrões de projeto GoF, foram conduzidos três experimentos controlados, conforme a metodologia apresentada na Seção 4.

[1]versão de 20/04/2025 - número de versão não disponibilizado pela ferramenta.

Para cada sugestão, as ferramentas apresentaram detalhes da aplicação contextual do padrão e seus benefícios associados [2]. Essas informações foram usadas como subsídio pelos especialistas para a avaliação das propostas sugeridas por cada ferramenta.

5.1 Experimento #1 - Agenda

Nesta seção, descreve-se o primeiro experimento elaborado com o intuito de avaliar a capacidade dos LLMs em sugerir padrões de projeto a partir de descrições de sistemas. O objetivo é verificar o grau de convergência entre as sugestões e analisar sua adequação ao problema proposto. No primeiro experimento, o contexto empregado foi o projeto de um aplicativo móvel para uma `Agenda de Compromissos`. A descrição utilizada neste e nos outros cenários reforça que o principal objetivo ao empregar padrões de projeto é melhorar a manutenibilidade para facilitar futuras modificações do software e obter escalabilidade.

A Tabela 3 apresenta um resumo das sugestões fornecidas pelos LLMs para o primeiro experimento, indicando, para cada padrão de projeto, sua respectiva categoria, uma breve descrição da aplicação sugerida e os principais benefícios associados. Observa-se uma clara convergência na recomendação do padrão comportamental *Observer*, indicado por todas as ferramentas avaliadas. O padrão criacional *Factory Method* foi sugerido por três LLMs, enquanto o padrão estrutural *Composite* apareceu em duas respostas. Já os padrões *Proxy* (estrutural), *State* e *Strategy* (comportamentais) foram mencionados por apenas um LLM cada.

Esses resultados iniciais demonstram não apenas a capacidade dos LLMs em identificar padrões clássicos aplicáveis ao problema, como também revelam um certo grau de convergência entre os modelos, especialmente em relação ao padrão *Observer*, frequentemente utilizado em aplicações com atualizações reativas de estado.

5.2 Experimento #2 – Ferramenta de CAD

O segundo experimento considerou o desenvolvimento de um software desktop voltado ao projeto de `Circuitos Lógicos`, oferecendo funcionalidades de edição e simulação. Esse tipo de aplicação se enquadra na categoria de ferramentas de *Computer-Aided Design* (CAD), ou projeto assistido por computador, comumente utilizadas para facilitar a criação, modificação e visualização de sistemas técnicos. Assim como no estudo anterior, o uso de padrões de projeto visa aprimorar a manutenibilidade da aplicação, facilitando futuras modificações e garantindo maior escalabilidade.

A Tabela 4 apresenta um resumo das sugestões fornecidas pelos LLMs para esse cenário, especificando, para cada padrão indicado, sua categoria, uma síntese de sua aplicação no contexto proposto e os benefícios esperados. Entre os padrões sugeridos, destaca-se o padrão estrutural *Composite*, indicado por todas as ferramentas. Os padrões comportamentais *Observer* e *Command* foram recomendados por dois LLMs cada, assim como o padrão criacional *Factory Method*. Outros padrões, como *Strategy* e *Builder*, foram sugeridos por apenas um LLM cada.

Esses resultados apontam para uma tendência dos LLMs em reconhecer padrões comumente aplicáveis em ferramentas gráficas

[2]As iterações com cada ferramenta poderão ser acessadas como recursos online posteriormente em caso de aceite do artigo.

Tabela 3: Experimento #1 - Análise comparativa das sugestões de Padrões de Projeto pelos LLMs.

Padrão Sugerido	Categoria	Aplicação Sugerida	Benefícios Apontados
Observer	Comportamental	**ChatGPT:** Implementar o sistema de notificações (alarmes) e também para a sincronização de agendas compartilhadas. **Gemini:** Sincronizar objetos dependentes (ex: Alarmes de um Evento) e atualizar múltiplas "visões"em tempo real (ex: UIs de Agendas compartilhadas). **Manus:** Ideal para lidar com a funcionalidade de alarmes que precisam se ajustar automaticamente quando um evento é editado. **DeepSeek:** Gerenciar notificações/alarmes e sua relação com eventos; Notificar usuários colaboradores sobre mudanças em agendas compartilhadas.	**ChatGPT:** Desacoplamento, fácil extensão de notificações. **Gemini:** Garantia de Consistência; Baixo Acoplamento; Escalabilidade. **Manus:** Manutenibilidade, Escalabilidade e Consistência de Dados. **DeepSeek:** Desacoplamento; Escalabilidade.
Factory Method	Criacional	**ChatGPT:** Pra a criação de Eventos, que podem ser de diferentes tipos: Evento diário (de dia inteiro), Evento simples (com hora de início e fim) e futuramente: evento recorrente, evento de múltiplos dia. **Gemini:** Criar diferentes tipos de "Eventos"(diário, simples, periódico) sem acoplar o código cliente às suas classes concretas. **DeepSeek:** Criar diferentes tipos de "eventos"(diários, simples, futuramente periódicos) sem acoplar o código à lógica de instanciação.	**ChatGPT:** Criação flexível, facilitando a adição de novos tipos de eventos no futuro. **Gemini:** Manutenibilidade; Baixo Acoplamento; Centralização da Lógica de Criação. **DeepSeek:** Facilita a adição de novos tipos de evento; Centraliza a lógica de criação; Alinha-se com o princípio "Open/Closed".
Composite	Estrutural	**ChatGPT:** Modelar a hierarquia de agendas e eventos, especialmente considerando que uma agenda contém múltiplos eventos, e futuramente eventos podem ter subeventos. **DeepSeek:** Modelar a hierarquia de agendas e eventos.	**ChatGPT:** Manipulação uniforme, escalável, preparado para expansão. **DeepSeek:** Simplifica a manipulação de estruturas complexas; Facilita a adição de novos tipos de eventos compostos sem refatorar o código existente.
Proxy	Estrutural	**Gemini:** Gerenciar permissões de acesso em objetos compartilhados, como uma Agenda onde colaboradores têm acesso limitado em comparação ao dono.	**Gemini:** Controle de Acesso Transparente; Separação de Responsabilidades; Flexibilidade.
State	Comportamental	**Manus:** Pode ser empregado para gerenciar o ciclo de vida dos convites para colaboração em agendas.	**Manus:** Manutenibilidade, Escalabilidade e Clareza do Código.
Strategy	Comportamental	**Manus:** Gerenciar os diferentes tipos de eventos (diário, simples, e futuramente, recorrente ou de múltiplos dias).	**Manus:** Manutenibilidade, Escalabilidade e Flexibilidade.

e interativas, reforçando a adequação de certos padrões ao tipo de funcionalidade esperada nesse domínio.

5.3 Experimento #3 – Ferramenta de Gestão de Projetos

Neste terceiro experimento, foi considerada a construção de uma ferramenta web para a `Gestão de Projetos de Software`. Assim como nos casos anteriores, o uso de padrões de projeto tem como objetivo aprimorar a manutenibilidade do sistema, facilitando futuras modificações e aumentando sua escalabilidade. A Tabela 5 resume os padrões indicados por cada LLM com base no prompt e na descrição textual da aplicação. Para cada padrão sugerido, são apresentados sua categoria, uma síntese da aplicação no contexto e os benefícios associados.

Os resultados revelam que os padrões *Composite* e *Observer* foram os mais frequentemente recomendados, aparecendo em quatro e três ferramentas, respectivamente. Os padrões *State* e *Factory Method* foram sugeridos por dois LLMs cada, enquanto o padrão *Strategy* foi citado apenas uma vez. A convergência nas recomendações para certos padrões pode indicar uma percepção consistente por parte dos LLMs sobre estruturas arquiteturais comuns em aplicações web colaborativas, como ferramentas de gestão.

Uma análise aprofundada dos três experimentos e a discussão sobre estes resultados são apresentados na próxima Seção 6.

6 Resultados

No intuito de avaliar os resultados obtidos, utilizou-se como referência o método de inspeção de software, tradicionalmente empregado para analisar a conformidade de artefatos de software. De acordo com Kalinowski [21], o processo clássico de inspeção envolve as seguintes etapas: planejamento da inspeção, revisão individual do artefato por cada participante, reunião em equipe para discussão e registro dos defeitos encontrados, encaminhamento desses defeitos ao autor do artefato para correção e, por fim, uma avaliação final para decidir sobre a necessidade de uma nova inspeção.

Para este experimento, o método de inspeção foi adaptado ao contexto específico da avaliação das sugestões geradas por LLMs. A adaptação visou manter os princípios fundamentais da inspeção (análise, registro estruturado e julgamento coletivo), porém ajustando-os à natureza das sugestões automáticas, que não são artefatos tradicionais de software, mas sim recomendações geradas por um modelo computacional.

Na etapa de validação, foram convidados especialistas com conhecimento técnico na área do projeto. Esses especialistas foram responsáveis por inspecionar as sugestões fornecidas pelas ferramentas, avaliando sua adequação ao contexto e aos seus requisitos. Nestes experimentos, contamos com duas pessoas especialistas, do gênero feminino, com 20 e 10 anos de experiência em projeto de software orientado a objetos. Nos casos em que houve divergência entre as avaliadoras, as discrepâncias foram solucionadas com a

Tabela 4: Experimento #2 - Análise comparativa das sugestões de Padrões de Projeto pelas LLMs.

Padrão Sugerido	Categoria	Aplicação Sugerida	Benefícios Apontados
Composite	Estrutural	**ChatGPT:** Modelagem dos circuitos hierárquicos, onde um circuito pode conter portas lógicas e subsistemas compostos, permitindo tratamento uniforme. **Gemini:** Unificar o tratamento de componentes simples (portas) e compostos (subsistemas) em uma estrutura hierárquica. **Manus:** Permite tratar portas lógicas e subsistemas de forma uniforme em uma hierarquia, facilitando operações recursivas como desenho e simulação de circuitos. **DeepSeek:** Modelar a hierarquia de componentes, tratando tanto componentes individuais (ex.: porta AND) quanto subsistemas como objetos do mesmo tipo, permitindo recursividade na composição.	**ChatGPT:** Manipulação uniforme, fácil expansão, simplifica recursão para tratar geração de código, simulação, etc. **Gemini:** Estrutura em Árvore Simplificada; Facilita a Hierarquia; Melhora a Manutenibilidade. **Manus:** Encapsula a complexidade da estrutura hierárquica; Facilita a adição de novos tipos de componentes; Simplifica o código. **DeepSeek:** Simplifica a manipulação de estruturas hierárquicas; Facilita a adição de novos tipos de componentes no futuro; Permite operações recursivas.
Observer	Comportamental	**Gemini:** Sincronizar a interface gráfica e a simulação com as mudanças de estado dos componentes (posição, valor) através de notificações. **DeepSeek:** Notificar automaticamente a interface gráfica sobre mudanças no circuito para atualizar a visualização e gerenciar a dependência entre o modelo (circuito) e as visualizações.	**Gemini:** Baixo Acoplamento; Sincronização Automática; Flexibilidade. **DeepSeek:** Desacopla a lógica do circuito da interface gráfica; Permite adicionar novos "observadores"; Garante consistência automática entre modelo e visualização.
Command	Comportamental	**ChatGPT:** Gerenciamento das ações do usuário na interface gráfica, como adicionar, editar, ou remover componente. Implementar facilmente as funcionalidades de Desfazer e Refazer. **Manus:** Encapsula ações de edição como objetos, permitindo execução, desfazer/refazer e facilitando histórico, macros e filas de comandos.	**ChatGPT:** Facilita a implementação robusta de undo/redo, aumentando muito a usabilidade. **Manus:** Suporte a implementação das funcionalidades desfazer/refazer; Desacopla quem solicita uma ação de quem a executa; Facilita a adição de novas operações.
Factory Method	Criacional	**ChatGPT:** Centraliza a criação dos componentes do circuito, sejam eles portas lógicas, conexões ou subsistemas compostos. **Gemini:** Centralizar a criação de componentes de circuito em uma classe "fábrica"para desacoplar a lógica de instanciação da interface gráfica.	**ChatGPT:** Organiza a criação dos objetos, melhorando a coesão e facilitando manutenção. **Gemini:** Centralização da Lógica de Criação; Maior Flexibilidade; e Redução do Acoplamento.
Strategy	Comportamental	**Manus:** Permite aplicar diferentes algoritmos de simulação ou geração de código VHDL;	**Manus:** Evita o uso de condicionais complexos. Melhora a manutenibilidade do código.
Builder	Criacional	**DeepSeek:** Construir subsistemas complexos passo a passo, especialmente durante a decomposição hierárquica e gerar automaticamente código VHDL a partir da estrutura do circuito.	**DeepSeek:** Isola a lógica de construção de subsistemas e geração de código, facilitando a manutenção; Permite variar a representação sem alterar a lógica do circuito; Útil para operações como exportação de imagens por nível hierárquico.

participação de uma terceira pessoa especialista, com experiência equivalente, que atuou como avaliadora de desempate para consolidar a decisão final.

Cada sugestão foi analisada individualmente, e os especialistas registraram suas avaliações buscando responder a seguinte pergunta: "O padrão de projeto sugerido se adequa devidamente ao contexto do projeto descrito?". As respostas foram então organizadas no seguinte formato:

- **Sim (s)**: quando a sugestão foi considerada **100% adequada** ao projeto;
- **Não (n)**: quando a sugestão foi considerada **inadequada**;
- **Parcial (p)**: quando a sugestão apresentava **adequação limitada**, ou seja, não traria benefícios diretos ao projeto ou não atendia plenamente aos requisitos não funcionais especificados.

Essas classificações permitiram uma análise sistemática da utilidade das recomendações das ferramentas para cada uma das aplicações empregadas nos experimentos.

A Tabela 6 apresenta a avaliação dos especialistas em relação aos padrões sugeridos no contexto do experimento 1. O padrão *Observer*, recomendado por todos os LLMs, foi considerado adequado pelos especialistas, pois permite a notificação automática de alterações em eventos, funcionalidade necessária quando há alarmes associados a esses eventos. O padrão *Factory Method*, sugerido por três LLMs, foi avaliado como parcialmente adequado. Apesar de poder facilitar a criação de objetos do tipo eventos a partir de uma interface comum, os especialistas identificaram ganhos limitados com sua aplicação no cenário descrito. O mesmo julgamento foi dado aos padrões *Proxy* e *State*. O primeiro poderia ser útil em cenários que exigissem controle de acesso ou intermediação no uso de recursos — aspectos não previstos na descrição do software. O segundo, embora aplicável em sistemas com transições explícitas de estado, não apresentou correspondência direta com funcionalidades da agenda ou dos eventos, sendo assim considerado de benefício limitado. Por outro lado, o padrão estrutural *Composite*, indicado por duas ferramentas, foi descartado. Embora agendas sejam compostas por eventos, a descrição do sistema não indicava a existência de subeventos ou hierarquias que justificassem o uso de uma estrutura composta. Por fim, o padrão *Strategy* foi considerado inadequado, pois não havia previsão de múltiplos algoritmos intercambiáveis para o tratamento de uma mesma operação, como seria exigido para justificar sua adoção.

Tabela 5: Experimento #3 - Análise comparativa das sugestões de Padrões de Projeto pelos LLMs.

Padrão Sugerido	Categoria	Aplicação Sugerida	Benefícios Apontados
Composite	Estrutural	**ChatGPT:** Representar a hierarquia de tarefas, especialmente para lidar com tarefas simples e tarefas complexas (que contêm subtarefas), permitindo tratamento uniforme. **Gemini:** Estruturar as tarefas de forma hierárquica, tratando tanto tarefas simples quanto complexas (compostas por outras tarefas) de maneira uniforme através de uma interface comum. **Manus:** Permite representar tarefas simples e complexas de forma hierárquica e uniforme, facilitando a manipulação e exibição da estrutura de tarefas do projeto **DeepSeek:** Modelar a hierarquia de tarefas (simples e complexas), onde tarefas complexas podem conter subtarefas (simples ou complexas) formando uma estrutura em árvore e permite tratar tarefas individuais e composições de tarefas de maneira uniforme.	**ChatGPT:** Melhora a legibilidade e a manutenibilidade da lógica relacionada a tarefas. **Gemini:** Simplificação do Código Cliente; Escalabilidade; e Estrutura em Árvore. **Manus:** Melhora a manutenibilidade; Aumenta a escalabilidade; Facilita o uso de recursão. **DeepSeek:** Simplifica o código; Facilita operações recursivas; Flexibilidade.
Observer	Comportamental	**Gemini:** Notificar objetos dependentes sobre mudanças de estado, permitindo ações como a remoção em cascata de tarefas ou a atualização de status de uma tarefa-mãe com base em suas filhas. **Manus:** Permite notificar automaticamente colaboradores sobre mudanças em projetos ou tarefas, mantendo os dados sincronizados sem acoplamento direto. **DeepSeek:** Notificar colaboradores sobre mudanças em tarefas alocadas a eles e pode ser usado para atualizar o status de uma tarefa complexa automaticamente quando todas as suas subtarefas são concluídas.	**Gemini:** Baixo Acoplamento; Escalabilidade; e Consistência de Dados. **Manus:** Promove o baixo acoplamento entre os objetos; Melhora a manutenibilidade; Aumenta a escalabilidade. **DeepSeek:** Desacoplamento entre os objetos;Permite adicionar novos observadores sem modificar a lógica principal; Melhora a responsabilidade única (SRP).
State	Comportamental	**ChatGPT:** Gerenciar as transições dos estados dos projetos e também das tarefas. **Gemini:** Encapsular os comportamentos de um Projeto que variam com seu estado em classes distintas, eliminando condicionais (if/else) da classe principal.	**ChatGPT:** Facilita a manutenção e a expansão das regras de negócio relacionadas a estados. Melhora a legibilidade do código. **Gemini:** Manutenibilidade; Princípio Aberto/Fechado; e Código Limpo.
Factory Method	Criacional	**ChatGPT:** Criação de objetos Projeto, TarefaSimples e TarefaComplexa, garantindo que cada tipo de objeto seja instanciado corretamente, **DeepSeek:** Útil para criar diferentes tipos de tarefas (simples ou complexas) sem expor a lógica de instanciação ao cliente e pode ser estendido para criar tarefas com configurações padrão.	**ChatGPT:** Facilita a adição de novos tipos de tarefas ou projetos no futuro. Aumenta a coesão e reduz o acoplamento entre classes. **DeepSeek:** Centraliza a criação de objetos, facilitando futuras modificações; Reduz a duplicação de código na criação de tarefas; Segue o princípio "Open/Closed".
Strategy	Comportamental	**Manus:** Permite gerenciar lógicas de status e regras de negócio de projetos e tarefas de forma flexível, isolando comportamentos em classes e evitando condicionais complexas.	**Manus:** Aumenta a flexibilidade, melhora a manutenibilidade e escalabilidade.

Tabela 6: Avaliação dos padrões sugeridos para o Experimento #1 - Agenda

Padrão sugerido	ChatGPT	Gemini	Manus	DeepSeek	Avaliação
Observer	X	X	X	X	S
Factory Method	X	X	–	X	P
Composite	X	–	–	X	N
Proxy	–	X	–	–	P
State	–	–	X	–	P
Strategy	–	–	X	–	N

No experimento 2, cujo contexto foi o desenvolvimento de uma ferramenta para edição e simulação de circuitos lógicos, seis padrões de projeto foram sugeridos pelos diferentes LLMs. A Tabela 7 resume essas sugestões e apresenta a avaliação dos especialistas quanto à adequação de cada padrão ao contexto do sistema. Como pode ser observado na Tabela, o padrão *Composite* foi o único sugerido de forma unânime e foi considerado totalmente aderente, uma vez que sua aplicação facilita a construção de hierarquias de componentes, característica essencial em sistemas que manipulam circuitos compostos por subsistemas.

Os padrões *Observer*, *Command* e *Factory Method* receberam duas indicações cada. Destes, o padrão *Observer* foi considerado adequado para este experimento. Neste contexto, entende-se que este padrão facilitará a implementação de atualizações nas conexões quando um componente for editado, reposicionado ou removido do projeto, impactando nas conexões que estejam ligadas a portas deste componente. O padrão *Command* foi considerado parcialmente aderente ao contexto do experimento. Tal padrão poderia auxiliar no gerenciamento das ações do usuário na interface gráfica, permitindo implementar facilmente as funcionalidades de desfazer e refazer. Embora estas funcionalidades sejam interessantes em software de edição gráfica, elas não haviam sido mencionadas na descrição do software, e por isso os especialistas avaliaram como parcial.

Já o padrão *Factory Method*, também duas indicações, não foi considerado necessário, já que a instância dos componentes a partir da barra de ferramentas não requer encapsulamento adicional da lógica de criação. Os padrões *Factory Method*, *Strategy* e *Builder*, cada um com apenas uma indicação, foram considerados inadequados

Tabela 7: Avaliação dos padrões sugeridos para o Experimento #2 - CAD

Padrão sugerido	ChatGPT	Gemini	Manus	DeepSeek	Avaliação
Composite	X	X	X	X	S
Observer	–	X	–	X	S
Command	X	–	X	–	P
Factory Method	X	X	–	–	N
Strategy	–	–	X	–	N
Builder	–	–	–	X	N

Tabela 8: Avaliação dos padrões sugeridos para o Experimento #3 - Gerência de projetos

Padrão sugerido	ChatGPT	Gemini	Manus	DeepSeek	Avaliação
Composite	X	X	X	X	S
Observer	–	X	X	X	S
State	X	X	–	–	P
Factory Method	X	–	–	X	N
Strategy	–	–	X	–	N

para este cenário. Já o padrão *Strategy* foi considerado inadequado, pois, embora tenha sido sugerido para permitir a alternância entre diferentes algoritmos de simulação ou geração de código VHDL, a descrição do sistema não contempla variações desse tipo. Dessa forma, não há um cenário claro no qual múltiplas estratégias precisem ser intercambiadas dinamicamente, como requer o uso efetivo desse padrão. Por fim, o padrão *Builder* também foi considerado inadequado, pois sua principal utilidade está na criação de objetos complexos passo a passo, o que não se aplica ao caso da ferramenta de CAD. Neste cenário, os componentes são instanciados de maneira direta e simples, a partir de interações com a interface, sem a necessidade de um processo de construção gradual ou personalizado.

Os resultados do terceiro experimento, detalhados na Tabela 8, indicam que dois padrões foram considerados totalmente adequados ao problema. O padrão *Composite*, com quatro indicações, foi a sugestão de maior consenso e, de forma similar ao experimento anterior, é ideal para tratar a decomposição de tarefas complexas em subtarefas. O *Observer*, com três indicações, também foi considerado uma solução apropriada para implementar notificações aos colaboradores sobre edições ou novas atribuições de tarefas. Em contrapartida, o padrão *State* (duas indicações) foi avaliado como parcialmente adequado. Embora a sugestão de gerenciar os estados de projetos e tarefas fosse pertinente, os especialistas ponderaram que a complexidade da lógica de transição não justificava a sobrecarga de classes adicionais que o padrão introduziria. Por fim, os padrões *Factory Method* e *Strategy* foram classificados como inadequados. Apesar de suas duas indicações, os especialistas não identificaram a necessidade de classes de fábrica de construtores distintos, proposta de aplicação sugerida para o *Factory Method*. Para o *Strategy* , sugerido apenas uma vez, não foram encontradas famílias de algoritmos intercambiáveis que justificassem sua aplicação no contexto do software.

Analisando os resultados dos três experimentos, observou-se que quando um padrão foi sugerido pelos quatro modelos, os especialistas concordaram que o emprego deste padrão seria benéfico ao projeto daquela aplicação. Quando foram sugeridos por três modelos, em dois dos projetos, a sugestão foi aprovada pelos especialistas, apenas em um dos experimentos foi considerada parcialmente benéfica. Já quando o padrão foi sugerido por dois modelos ou por um modelo apenas, sua adequação não é garantida. Nos experimentos realizados, nesses casos, a classificação variou entre as três opções: adequado, parcialmente e não adequado.

Tabela 9: Distribuição dos padrões de projeto sugeridos por LLMs nos três experimentos realizados

Padrão sugerido	ChatGPT	Gemini	Manus	DeepSeek	Total
Composite	3	2	2	3	10
Observer	1	3	2	3	9
Factory Method	3	2	0	2	7
State	1	1	1	0	3
Strategy	0	0	3	0	3
Command	1	0	1	0	2
Builder	0	0	0	1	1
Proxy	0	1	0	0	1

A Tabela 9 apresenta a frequência com que cada padrão de projeto foi sugerido pelos diferentes modelos nos três experimentos realizados. Ao comparar as respostas dos LLMs com os padrões mais utilizados em repositórios do GitHub, segundo Asaad and Avksentieva [6], verifica-se que a maioria dos padrões sugeridos encontra-se dentre aqueles mais empregados na área de desenvolvimento de software. Entre os padrões mais recorrentes em nossos experimentos estão *Composite* e *Factory Method*, ambos citados por ao menos um dos modelos nos três experimentos realizados. Embora não estejam no topo da lista dos mais utilizados em repositórios GitHub [6], esses padrões ainda figuram entre os mais adotados na prática e desempenham um papel fundamental na estruturação de sistemas e na criação de objetos, sendo especialmente relevantes do ponto de vista do design [7]. O padrão *Observer* e o *Strategy* estão entre os padrões mais utilizados em projetos reais e foram indicados por ao menos um dos modelos nos três experimentos realizados. Em especial, *Observer* e *Composite* foram mencionados pelos quatro LLMs, o que reforça sua popularidade e aplicabilidade. Os padrões *State*, *Command* e *Builder* estão entre os mais utilizados em projetos reais, e também aparecem entre os padrões sugeridos pelos modelos, ainda que com menor frequência.

O padrão *Proxy* foi sugerido apenas uma vez, refletindo sua menor popularidade em projetos reais. O estudo de Asaad and Avksentieva [6] considera os padrões como *Memento*, *Flyweight* e *Interpreter*, com baixa adoção na prática. Vale destacar que estes padrões não foram indicados por nenhum dos modelos em nenhum dos experimentos. De modo geral, os resultados sugerem que os LLMs analisados tendem a priorizar padrões amplamente adotados na prática, indicando um bom alinhamento entre suas recomendações e os cenários reais de desenvolvimento de software.

7 Limitações, Fraquezas e Ameaças à Validade

Nesta seção, discutimos as principais ameaças à validade de nosso estudo, organizadas em três categorias principais.

Ameaças à Validade de Constructo: Esta ameaça se refere à relação entre a teoria e a observação. Em nosso estudo, a principal ameaça é a avaliação da "adequação"de um padrão, que foi baseada

no julgamento de três especialistas. Embora sejam experientes, a avaliação é inerentemente subjetiva e pode não representar um consenso universal. Ainda, a descrição textual dos problemas, embora baseada em cenários reais de ensino, também pode não capturar toda a complexidade de um requisito industrial, influenciando as sugestões.

Ameaças à Validade Interna: Fatores internos podem ter influenciado nossos resultados. O design do prompt adotado (ver Seção 4) é uma variável crucial; diferentes prompts poderiam levar a resultados distintos. Além disso, o não-determinismo inerente aos LLMs significa que a repetição do mesmo experimento pode não gerar respostas idênticas, embora tenhamos mitigado isso salvando a primeira sugestão de cada ferramenta. Ressalta-se que, embora essa característica possa ser considerada benéfica em contextos criativos, tal análise está fora do escopo deste estudo, cujo foco é avaliar a adequação técnica das sugestões fornecidas.

Ameaças à Validade Externa: A generalização de nossos resultados é limitada. O estudo se restringe a três cenários de software e quatro LLMs específicos. Os resultados podem não ser os mesmos para outros tipos de problemas de software ou para versões futuras desses mesmos LLMs. Além disso, a análise foca exclusivamente no paradigma orientado a objetos, e as generalizações de aplicação a outros paradigmas não foram previstas.

8 Considerações Finais

As decisões de projeto impactam diretamente na qualidade e na manutenibilidade do software. Este artigo investigou a capacidade de diferentes LLMs atuarem como ferramentas de apoio a projetistas na escolha de padrões de projeto GoF, um desafio que exige a interpretação de descrições textuais, muitas vezes ambíguas, de um sistema.

As contribuições deste trabalho podem ser sumarizadas para dois públicos distintos. Para pesquisadores, este estudo oferece uma metodologia de avaliação e resultados de baseline para uma área emergente, servindo como ponto de partida para investigações futuras sobre a interação entre LLMs e o design de software. Os dados coletados, incluindo os prompts, as descrições e as saídas das ferramentas (disponibilizados como recursos online), representam um artefato valioso para a reprodutibilidade e a expansão desta pesquisa. Como a pesquisa foi realizada por profissionais atuantes na pesquisa e na docência, há o interesse de aproximação entre pesquisa e o ensino de Engenharia de Software, além de ampliar a avaliação e discussão dos resultados desta pesquisa, a ser feito em trabalhos futuros. Para profissionais da indústria, nosso trabalho fornece evidências empíricas de que LLMs podem atuar como ferramentas de brainstorming e suporte à decisão. A principal implicação prática é a estratégia de usar múltiplas ferramentas em conjunto: o consenso entre diferentes LLMs demonstrou ser um forte indicador da adequação de uma sugestão, oferecendo um filtro em potencial para arquitetos e projetistas de software no dia a dia.

Assim, a principal contribuição deste trabalho é a investigação da capacidade de múltiplos LLMs em sugerir padrões de projeto GoF a partir de descrições textuais, por vezes ambíguas, de um sistema. Através de três experimentos controlados distintos, com análise de especialistas, os resultados demonstraram uma forte correlação: sugestões unânimes (apontadas pelos quatro LLMs) foram consistentemente consideradas adequadas, enquanto a confiança na sugestão diminui progressivamente com um menor número de indicações. Os resultados apontaram para uma correlação positiva entre o consenso de sugestões dos modelos e a adequação percebida por especialistas. De modo geral, quanto maior o número de indicações concordantes entre os modelos, maior é a probabilidade de o padrão ser considerado apropriado. Isso sugere que a 'sabedoria coletiva' de várias ferramentas pode servir como um valioso filtro de confiança para os projetistas. Por outro lado, um número reduzido de indicações tende a refletir menor confiabilidade na recomendação. Adicionalmente, o modelo de prompt estruturado que utilizamos mostrou-se eficaz para facilitar a análise crítica humana, reforçando a importância da engenharia de prompt para extrair resultados úteis.

Como é apontado por inúmeros trabalhos, padrões de projeto estão relacionados diretamente à qualidade do software. Para trabalhos futuros, planejamos abordar as limitações discutidas na Seção 7 em duas frentes principais. A primeira diz respeito à exploração de modelos abertos e variação de parâmetros: pretendemos realizar novos experimentos com LLMs de código aberto, o que nos permitirá investigar sistematicamente o impacto de diferentes configurações – um aspecto ainda não explorado neste estudo – sobre a qualidade e consistência das sugestões. A segunda envolve o aprofundamento da avaliação da qualidade: propomos implementar as soluções de software indicadas, com o objetivo de mensurar quantitativamente, por meio de métricas consolidadas na literatura, o efeito dos padrões propostos sobre o código-fonte. Essa análise complementará e aprofundará a discussão apresentada neste trabalho. Desta forma, a distância entre a informação dada e a informação requerida aos modelos pode ser reduzida, aprimorando o estudo.

DISPONIBILIDADE DE ARTEFATOS

As descrições das aplicações utilizadas como entradas nos experimentos e as saídas geradas por cada ferramenta para cada descrição estão disponibilizadas em: https://drive.google.com/drive/folders/197CakVspz0PZNJsGh2DLOjLXBEVQnpRA?usp=drive_link.

AGRADECIMENTOS

Este trabalho foi financiado em parte pela Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS) e pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES).

REFERÊNCIAS

[1] Charu C. Aggarwal. 2023. *Neural Networks and Deep Learning: A Textbook* (2nd ed.). Springer Publishing Company, Incorporated.
[2] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. 1977. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, New York, USA. 1216 pages.
[3] Amazon Web Services. 2025. *Amazon Q Developer.* https://aws.amazon.com/pt/q/developer/
[4] Anthropic. 2024. *Claude 3.5 Sonnet Model Card Addendum.* Technical Report. Anthropic.
[5] Anysphere Inc. 2025. *Cursor.* https://cursor.com/
[6] Jameleh Asaad and Elena Avksentieva. 2025. Assessing the Impact of Gof Design Patterns on Software Engineering Practices. In *2025 International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM).* 930–934. doi:10.1109/ICIEAM65163.2025.11028537
[7] B. Bafandeh Mayvan, A. Rasoolzadegan, and Z. Ghavidel Yazdi. 2017. The state of the art on design patterns: A systematic mapping of the literature. *Journal of Systems and Software* 125 (2017), 93–118. doi:10.1016/j.jss.2016.11.030

[8] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609* (2023).

[9] Arifa I. Champa, Md Fazle Rabbi, Costain Nachuma, and Minhaz F. Zibran. 2024. ChatGPT in Action: Analyzing Its Use in Software Development. In *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. 182–186.

[10] Hai Dang, Ben Lafreniere, Tovi Grossman, Kashyap Todi, and Michelle Li. 2025. Authoring LLM-Based Assistance for Real-World Contexts and Tasks. In *Proceedings of the 30th International Conference on Intelligent User Interfaces (IUI '25)*. Association for Computing Machinery, New York, NY, USA, 211–230. doi:10.1145/3708359.3712164

[11] DeepSeek AI. 2025. *DeepSeek*. https://deepseek.com/

[12] Rushali Deshmukh, Rutuj Raut, Mayur Bhavsar, Sanika Gurav, and Yash Patil. 2025. Optimizing Human-AI Interaction: Innovations in Prompt Engineering. In *2025 3rd International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*. 1240–1246. doi:10.1109/IDCIOT64235.2025.10914815

[13] Hao Ding, Ziwei Fan, Ingo Guehring, Gaurav Gupta, Wooseok Ha, Jun Huan, Linbo Liu, Behrooz Omidvar-Tehrani, Shiqi Wang, and Hao Zhou. 2024. Reasoning and Planning with Large Language Models in Code Development. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Barcelona, Spain) *(KDD '24)*. Association for Computing Machinery, New York, NY, USA, 6480–6490. doi:10.1145/3637528.3671452

[14] Willian Freire, Murilo Boccardo, Daniel Nouchi, Aline Amaral, Silvia Vergilio, Thiago Ferreira, and Thelma Colanzi. 2024. Large Language Model-based suggestion of objective functions for search-based Product Line Architecture design. In *Anais do XVIII Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software* (Curitiba/PR). SBC, Porto Alegre, RS, Brasil, 21–30. doi:10.5753/sbcars.2024.3833

[15] Erich Gamma. 1995. *Design patterns: elements of reusable object-oriented software*. Pearson Education India.

[16] Erich Gamma. 2011. Design patterns—ten years later. *Software pioneers: contributions to software engineering* (2011), 688–700.

[17] Github/OpenAI. 2022. Github Copilot. Retrieved July 6, 2025 from https://github.com/features/copilot

[18] Google DeepMind. 2023. *Gemini*. https://deepmind.com/gemini

[19] Jasmin Jahić and Ashkan Sami. 2024. State of Practice: LLMs in Software Engineering and Software Architecture. In *2024 IEEE 21st International Conference on Software Architecture Companion (ICSA-C)*. 311–318. doi:10.1109/ICSA-C63560.2024.00059

[20] Hamraz Javaheri, Omid Ghamarnejad, Paul Lukowicz, Gregor Alexander Stavrou, and Jakob Karolus. 2024. ARAS: LLM-Supported Augmented Reality Assistance System for Pancreatic Surgery. In *Companion of the 2024 on ACM International Joint Conference on Pervasive and Ubiquitous Computing* (Melbourne VIC, Australia) *(UbiComp '24)*. Association for Computing Machinery, New York, NY, USA, 176–180. doi:10.1145/3675094.3677543

[21] Marcos Kalinowski. 2008. Introdução à inspeção de software. *Revista Engenharia de Software: Qualidade de software* 1 (2008), 68–74.

[22] Sascha Kaltenpoth and Oliver Müller. 2025. Don't Touch the Power Line - A Proof-of-Concept for Aligned LLM-Based Assistance Systems to Support the Maintenance in the Electricity Distribution System. *SIGENERGY Energy Inform. Rev.* 4, 4 (Feb. 2025), 16–22. doi:10.1145/3717413.3717415

[23] Rui Mao, Guanyi Chen, Xulang Zhang, Frank Guerin, and Erik Cambria. 2023. GPTEval: A survey on assessments of ChatGPT and GPT-4. *arXiv preprint arXiv:2308.12488* (2023).

[24] Ggaliwango Marvin, Nakayiza Hellen, Daudi Jjingo, and Joyce Nakatumba-Nabende. 2024. Prompt Engineering in Large Language Models. In *Data Intelligence and Cognitive Informatics*, I. Jeena Jacob, Selwyn Piramuthu, and Przemyslaw Falkowski-Gilski (Eds.). Springer Nature Singapore, Singapore, 387–402.

[25] Timothy R McIntosh, Teo Susnjak, Tong Liu, Paul Watters, and Malka N Halgamuge. 2023. From google gemini to openai q*(q-star): A survey of reshaping the generative artificial intelligence (ai) research landscape. *arXiv preprint arXiv:2312.10868* (2023).

[26] Monica (Butterfly Effect AI). 2025. *Manus AI*. https://manus.im/

[27] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2025. A Comprehensive Overview of Large Language Models. *ACM Trans. Intell. Syst. Technol.* (June 2025). doi:10.1145/3744746 Just Accepted.

[28] Fnu Neha and Deepshikha Bhati. 2025. A Survey of DeepSeek Models. *Authorea Preprints* (2025).

[29] OpenAI. 2023. *ChatGPT*. https://www.openai.com/chatgpt

[30] Zhenyu Pan, Xuefeng Song, Yunkun Wang, Rongyu Cao, Binhua Li, Yongbin Li, and Han Liu. 2025. Do Code LLMs Understand Design Patterns? arXiv:2501.04835 [cs.SE] https://arxiv.org/abs/2501.04835

[31] Sushant Kumar Pandey, Sivajeet Chand, Jennifer Horkoff, Miroslaw Staron, Miroslaw Ochodek, and Darko Durisic. 2025. Design pattern recognition: a study of large language models. *Empirical Software Engineering* 30, 3 (2025), 69.

[32] Larissa Schmid, Tobias Hey, Martin Armbruster, Sophie Corallo, Dominik Fuchß, Jan Keim, Haoyu Liu, and Anne Koziolek. 2025. Software Architecture Meets LLMs: A Systematic Literature Review. arXiv:2505.16697 [cs.SE] https://arxiv.org/abs/2505.16697

[33] Minjie Shen and Qikai Yang. 2025. From mind to machine: The rise of manus ai as a fully autonomous digital agent. *arXiv preprint arXiv:2505.02024* (2025).

[34] Siqi Shen, Lajanugen Logeswaran, Moontae Lee, Honglak Lee, Soujanya Poria, and Rada Mihalcea. 2024. Understanding the capabilities and limitations of large language models for cultural commonsense. *arXiv preprint arXiv:2405.04655* (2024).

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[36] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C. Schmidt. 2024. *ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design*. Springer Nature Switzerland, Cham, 71–108. doi:10.1007/978-3-031-55642-5_4

[37] Zihao Yi, Jiarui Ouyang, Yuwen Liu, Tianhao Liao, Zhe Xu, and Ying Shen. 2024. A Survey on Recent Advances in LLM-Based Multi-turn Dialogue Systems. arXiv:2402.18013 [cs.CL] https://arxiv.org/abs/2402.18013

[38] Quanjun Zhang, Chunrong Fang, Yang Xie, Yaxin Zhang, Yun Yang, Weisong Sun, Shengcheng Yu, and Zhenyu Chen. 2024. A Survey on Large Language Models for Software Engineering. arXiv:2312.15223 [cs.SE] https://arxiv.org/abs/2312.15223

[39] Xun Zhang, Hironori Washizaki, Nobukazu Yoshioka, and Yoshiaki Fukazawa. 2022. Detecting Design Patterns in UML Class Diagram Images using Deep Learning. In *2022 IEEE/ACIS 23rd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*. 27–32. doi:10.1109/SNPD54884.2022.10051795