Análise Arquitetural do crawl4ai

Um Comparativo: Modelos de IA vs. Análise Humana

Baseado no estudo de Carlos Gois, Felipe Moura, João Arruda, Nicolas Jesus, Samuel Pinho, Vinicius Micska, e Vitor Lima.

O Desafio

O objetivo deste estudo foi investigar e comparar a eficácia de diferentes estratégias na identificação de padrões arquiteturais em um projeto de software real. Colocamos Modelos de Linguagem (LLMs) de ponta contra a análise manual de engenheiros.

O Objeto de Estudo: Projeto crawl4ai

O `crawl4ai` é um web scraper avançado, projetado para realizar buscas profundas e otimizadas em múltiplos sites, retornando um texto limpo em formato markdown para o treinamento eficiente de LLMs. Sua natureza emergente o torna um alvo perfeito para análise arquitetural.

Os "Detetives": Quem Analisou o Código?

Modelos de IA (LLMs)

- Qwen2.5-Coder-7B-Instruct
- Qwen2.5-Coder-0.5B-Instruct
- CodeLlama-7b-hf
- CodeBERT-base (para clusterização)

Analistas Manuais

- Nicolas Matheus Ferreira de Jesus
- Vinicius Vasconi Villas Boas Micska

Como Eles Investigaram?

Estratégias dos LLMs

Análise de Issues (CodeLlama)
Agrupamento temático e síntese em fases para evitar estouro de memória.

Análise de Cód. Fonte (Qwen, CodeLlama)
Múltiplas táticas: prompt único com arquivos-chave, análise por arquivo, e chunking para modelos menores.

Clusterização Semântica (CodeBERT)
Vetorização de arquivos para agrupar módulos por função, guiando a análise do CodeLlama.

Estratégias Manuais

Leitura de Código e Lógica (N. Ferreira)
Análise do README, seguida por uma leitura profunda do núcleo (`asyncwebcrawler`) e suas dependências.

Análise de Dependências (V. Micska)
Inspeção de `requirements.txt` e `Dockerfile` (FastAPI, aioSQLite, REDIS) para inferir padrões de alto nível.

Mergulho Profundo nas Metodologias

Processo: Análise Manual por Inferência (V. Micska)

`aioSQLite` (BD Assíncrono)	→	Arquitetura **Event Driven**
`FastAPI` (Servidor ASGI)	→	Arquitetura **Microserviços**
`PDF Parsing` (Manipulação)	→	Arquitetura **Pipe & Filter**
`REDIS` (BD Local)	→	Padrão **Proxy (Cache)**

Processo: Análise LLM em 2 Fases (S. Pinho)

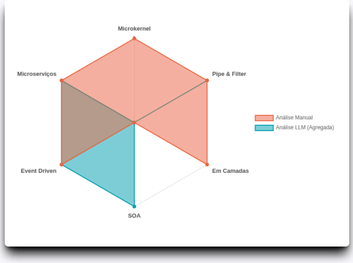
FASE 1: CodeBERT (Segmentação Semântica)
Vetorização de arquivos e clusterização K-Means para encontrar os módulos funcionais mais relevantes (Clusters 3 e 4).

←

FASE 2: CodeLlama (Auditoria de Código)
Análise focada nos arquivos dos Clusters 3 e 4 para identificar padrões de projeto e arquiteturais com base em evidências de código.

Descobertas: Padrões Arquiteturais

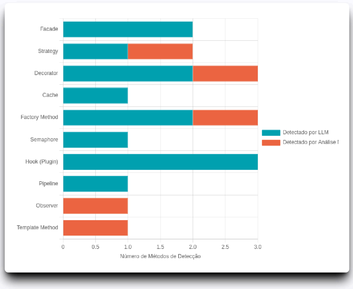
A análise revelou uma visão diferente da arquitetura dependendo do método. O radar compara a frequência com que cada método identificou os principais estilos arquiteturais.



Análise: A Análise Manual (N. Ferreira) foi a única a identificar os padrões `Microkernel` e `Em Camadas`, sugerindo uma melhor compreensão holística da estrutura central do app. Os LLMs foram mais fortes na detecção de SOA.

Descobertas: Padrões de Projeto (Nível de Código)

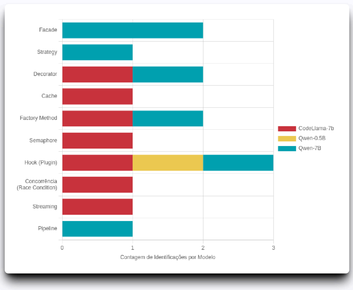
No nível do código, a colaboração e as sobreposições tornam-se claras. Este gráfico mostra quais padrões foram detectados e por quem.



Análise: Padrões como `Strategy`, `Decorator` e `Factory` foram encontrados por ambos, mostrando sobreposição. No entanto, os LLMs sozinhos encontraram padrões de concorrência (`Semaphore`, `Cache`), enquanto os humanos encontraram padrões clássicos de OO (`Observer`, `Template Method`).

Atribuição de Descoberta por Modelo de IA

Este gráfico detalha quais modelos de IA foram responsáveis por identificar cada padrão de projeto. O padrão "Hook (Plugin)" foi o mais consensual entre os modelos.



Análise: O `CodeLlama` foi particularmente eficaz na identificação de padrões de concorrência e streaming. O `Qwen-7B` (utilizado por David, Felipe e João) foi forte em padrões estruturais como `Facade` e `Pipeline`. O `Qwen-0.5B`, mais leve, só identificou o padrão Hook.

O Veredito: Prós e Contras de Cada Método

Qwen 0.5B (Leve)

Forte: Excelente para análise de alto nível (README, estrutura de pastas).
Fraco: Incapaz de detectar padrões complexos no código-fonte.

CodeLlama-7b

Forte: Detectou padrões de código específicos (Cache, Semaphore, Concorrência).
LIMITAÇÃO CRÍTICA (V. Leonardo): Sofreu de "Alucinação Persistente" (afirmou Singleton sem fatos) e "Inconsistência" (ignorou fatos que ele mesmo encontrou).

Qwen 7B (Pesado)

Forte: Poderoso para síntese e análise por arquivo.
Fraco: Muito dependente da engenharia de prompt; "assume" padrões sem evidência clara.

CodeBERT-base

Forte: Ótimo para clusterização semântica de arquivos, separando "núcleo" de "implementações".
Fraco: Não interpreta; apenas agrupa.

Análise Manual

Forte: Identificação holística imbatível; conectou dependências (FastAPI, REDIS) à lógica de código para identificar a arquitetura híbrida (Microkernel + Pipe & Filter).
Fraco: Mais lento; pode perder padrões de implementação de nicho (ex: concorrência) que os LLMs detectaram.

Conclusão

Arquitetura Final do crawl4ai

Híbrida

Microkernel + Pipe & Filter

Uma combinação que promove extensibilidade (plugins) e processamento de dados sequencial e assíncrono.

Principais Lições Aprendidas

- **LLMs Têm Limites:** Modelos leves falham em profundidade. Modelos pesados correm o risco de "alucinar" e requerem prompts complexos para mitigar isso.
- **O Fator Humano é Chave:** A análise manual foi a estratégia mais robusta para sintetizar a visão geral do sistema, conectando dependências externas com a lógica do núcleo.

Infográfico gerado a partir do relatório de Engenharia de Software II da Universidade Federal de Sergipe (08/11/2025).