UNIVERSIDADE FEDERAL DE MINAS GERAIS
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Arquitetura de Inteligência Artificial Generativa na Nuvem para Automação Documental: Estudo no Sistema socket.cc

Relatório Final — Projeto Orientado em Computação II

Aluno: Arthur Linhares Madureira

Orientador: Prof. Eduardo Figueiredo

Belo Horizonte, junho de 2026

Resumo

Este trabalho apresenta o desenvolvimento da segunda fase do sistema *socket.cc*, uma plataforma web concebida para digitalizar e centralizar o processo de submissão e aceite de orientações de trabalhos de conclusão de curso no Departamento de Ciência da Computação da UFMG. Na primeira fase (POC I), foram implementados os módulos fundamentais da aplicação, incluindo gerenciamento de perfis de usuário, upload de documentos PDF para o Amazon S3 e persistência de dados com SQL via Prisma ORM. Nesta segunda fase (POC II), o sistema foi evoluído de um repositório passivo para uma plataforma de inteligência documental, integrando a API Google Gemini para análise automática de propostas acadêmicas em PDF. A contribuição central é um fluxo de análise em três etapas — formulário, análise por IA e exibição de *feedback* estruturado — que fornece sugestões construtivas ao aluno antes da confirmação do envio. Complementarmente, foi conduzido um estudo experimental comparando três técnicas de *prompt engineering* — Zero-shot, Few-shot e Chain-of-Prompts — aplicadas a 15 documentos PDF de qualidade variável, com o objetivo de identificar a estratégia mais eficaz para avaliação de propostas acadêmicas. O sistema foi implantado em infraestrutura AWS (EC2 com IP elástico e S3), com pipeline serverless via AWS Lambda para processamento de PDF.

Palavras-chave: Inteligência Artificial Generativa; *Prompt Engineering*; Computação em Nuvem; AWS; Google Gemini; Vue.js; Node.js; Automação Documental.

Sumário

1 Introdução

1.1 Contextualização e Motivação

O processo de orientação de trabalhos de conclusão de curso em departamentos de Computação envolve uma série de etapas administrativas que, historicamente, são conduzidas de forma fragmentada: trocas de e-mails dispersas, conversas informais em corredores e preenchimento manual de formulários. Essa dinâmica dificulta o acompanhamento sistemático do fluxo de candidaturas tanto pelos professores quanto pelos alunos, gerando ineficiência e perda de informações.

O *socket.cc* surgiu como resposta a esse problema. O nome estabelece uma metáfora técnica com os *sockets* de comunicação em redes: o sistema funciona como ponto de extremidade que conecta de forma direta e padronizada a demanda do "cliente" (o aluno com uma proposta de pesquisa) à oferta do "servidor" (o professor com disponibilidade de orientação), operando como um *marketplace* acadêmico.

Na fase anterior (POC I), foram construídos os alicerces técnicos do sistema: autenticação via JWT, gerenciamento de perfis de alunos e professores, upload seguro de documentos para o Amazon S3 e um banco de dados relacional. Entretanto, o sistema comportava-se como um repositório passivo — os documentos PDF submetidos eram armazenados sem qualquer processamento de seu conteúdo, exigindo que o professor realizasse o download e a leitura integral de cada arquivo para avaliar a viabilidade de orientação.

1.2 Objetivos

O objetivo geral do POC II é transformar o *socket.cc* de um repositório passivo em uma plataforma de inteligência documental, capaz de analisar automaticamente o conteúdo das propostas submetidas e fornecer valor imediato tanto aos alunos (antes do envio) quanto aos professores (após a submissão).

Os objetivos específicos são:

1. Integrar a API Google Gemini ao *backend* para análise semântica de PDFs acadêmicos em língua portuguesa brasileira;
2. Implementar um fluxo de análise pré-submissão com *feedback* estruturado, permitindo ao aluno corrigir sua proposta antes de enviá-la ao professor;
3. Desenvolver uma pipeline serverless na AWS para processamento pós-submissão, gerando resumos executivos e etiquetas temáticas automaticamente;

4. Conduzir um estudo experimental comparativo de três técnicas de *prompt engineering* (Zero-shot, Few-shot e Chain-of-Prompts), avaliando qual estratégia produz avaliações mais precisas de propostas acadêmicas;

5. Implantar o sistema em infraestrutura AWS de produção com alta disponibilidade.

2 Referencial Teórico

2.1 Modelos de Linguagem de Grande Escala (LLMs)

Modelos de Linguagem de Grande Escala (*Large Language Models* — LLMs) são sistemas de IA treinados em corpora massivos de texto, capazes de realizar tarefas de compreensão e geração de linguagem natural sem necessidade de treinamento específico por tarefa. A família Gemini representa um LLM multimodal capaz de processar simultaneamente texto e documentos PDF, o que é particularmente relevante para o contexto deste trabalho.

2.2 Prompt Engineering

Prompt engineering é a prática de elaborar entradas textuais (*prompts*) para LLMs de forma a maximizar a qualidade e relevância das saídas geradas. Três técnicas são centrais para este trabalho:

- **Zero-shot Prompting**: instrução direta ao modelo sem exemplos prévios. Avalia a capacidade de generalização do LLM;
- **Few-shot Prompting**: fornecimento de exemplos de entrada-saída desejados no próprio *prompt*, orientando o tom e o formato esperado da resposta;
- **Chain-of-Prompts / Chain-of-Thought**: decomposição do problema em múltiplas chamadas sequenciais ao modelo, onde cada etapa enriquece o contexto da próxima. Diferentemente do *Chain-of-Thought* convencional (que ocorre em uma única chamada), esta variante distribui o raciocínio em chamadas independentes.

2.3 Arquiteturas Serverless e Event-Driven

O modelo serverless permite a execução de funções sob demanda sem gerenciamento explícito de servidores. A AWS Lambda executa funções em resposta a eventos, como a criação de objetos no Amazon S3 (evento `S3:ObjectCreated`), viabilizando pipelines de processamento assíncrono sem custo contínuo de infraestrutura. Esse paradigma é especialmente adequado para tarefas de processamento documental de latência tolerável, como a geração de resumos por IA após a submissão de uma proposta.

3 Contribuição: Arquitetura e Implementação

3.1 Visão Geral da Arquitetura

A Figura 1 ilustra a arquitetura completa do sistema *socket.cc* após as evoluções do POC II. O sistema é dividido em três camadas principais:

1. **Frontend (Vue.js 3 + Vite)**: SPA que gerencia o fluxo de interação do usuário, incluindo o novo fluxo de análise pré-submissão;
2. **Backend (Node.js/Express)**: API REST que orquestra as operações de negócio, persiste dados via Prisma ORM e integra a API Gemini;
3. **Infraestrutura AWS**: EC2 (servidor de produção), S3 (armazenamento de PDFs) e Lambda (processamento pós-submissão).

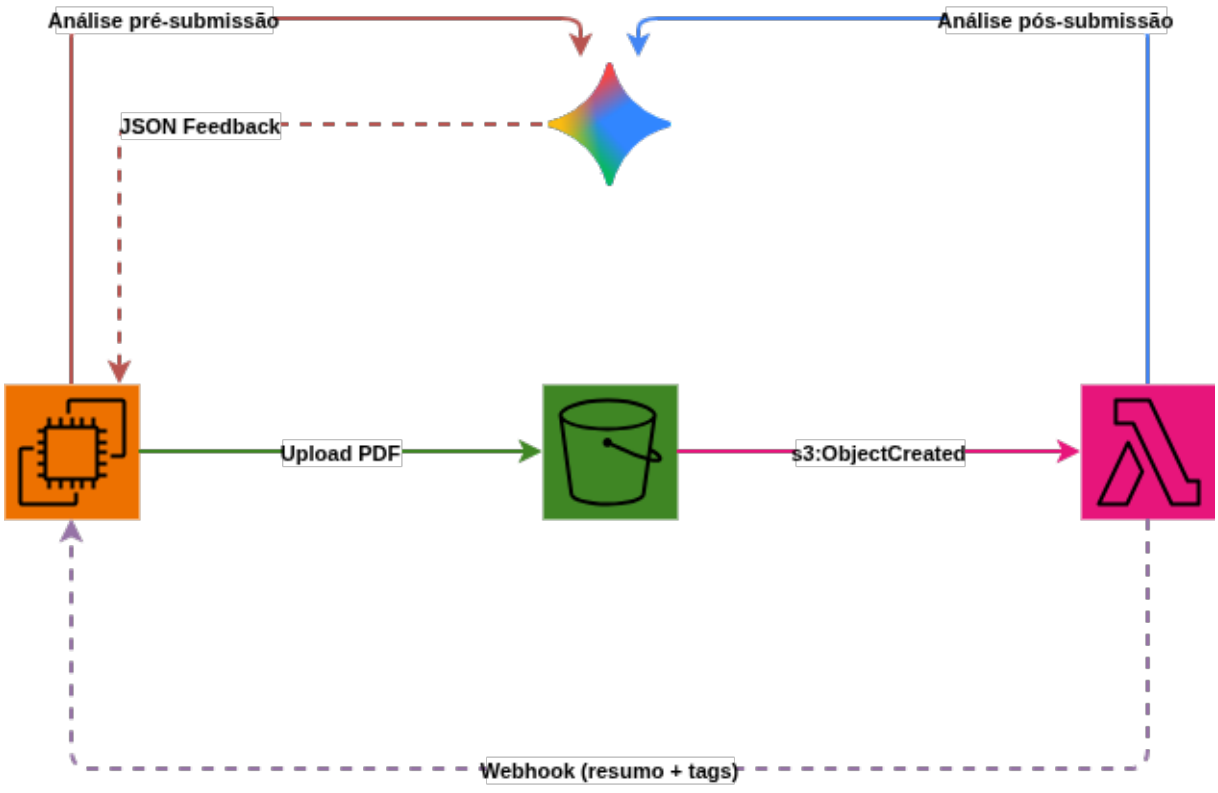

Figura 1: Arquitetura geral do sistema *socket.cc* após POC II.

3.2 Decisão de Design: Google Gemini vs. AWS Textract + Bedrock

A proposta original do POC II previa o uso de AWS Textract para extração estruturada de texto dos PDFs e do AWS Bedrock como interface unificada para inferência de LLMs.

Entretanto, durante a fase de implementação, verificou-se que **ambos os serviços não estavam disponíveis na conta AWS utilizada no projeto**.

O AWS Bedrock, em particular, exige uma solicitação formal de acesso (*model access request*) que precisa ser aprovada pela AWS para cada modelo de fundação individualmente. Ao tentar habilitar os modelos necessários (Claude, Titan e Llama) no console da AWS, a solicitação permaneceu pendente por período incompatível com o cronograma do trabalho.

Diante desse cenário, optou-se pelo **Google Gemini** como substituto funcional.

A substituição não alterou a arquitetura lógica do sistema: a separação entre análise pré-submissão (síncrona, Gemini via API do backend) e processamento pós-submissão (assíncrono, Gemini via AWS Lambda) foi preservada, mantendo a filosofia *event-driven* proposta originalmente. O AWS Lambda e o Amazon S3 foram utilizados conforme planejado, pois esses serviços estavam disponíveis no Free Tier sem restrições adicionais.

3.3 Módulo de Análise Pré-Submissão

3.3.1 Backend: serviço de análise

O método `analyzeProposal` foi adicionado à classe `ProposalService`. Ele recebe o buffer do PDF enviado pelo aluno, constrói um *prompt* estruturado em português brasileiro e invoca a API Gemini, solicitando uma resposta estritamente em formato JSON com cinco categorias de avaliação.

Listing 1: Método `analyzeProposal` em `proposalService.js`.

```
async analyzeProposal(pdfBuffer) {
    const genAI = new GoogleGenerativeAI(process.env.GEMINI_API_KEY);
    const model = genAI.getGenerativeModel({ model: 'gemini-3-flash-
       preview' });

    const prompt = `Voce e um revisor academico especializado. Analise
       este
PDF de proposta academica e forneca feedback construtivo em portugues.
Avalie: ortografia, clareza, estrutura, qualidade tecnica e normas
   academicas.
Responda APENAS em JSON valido:
{
  "pontuacao": <1-10>,
  "resumo_geral": "<2-3 frases>",
  "dicas": [
    { "categoria": "...", "nivel": "<alta|media|baixa>",
      "sugestoes": ["..."] }
  ]
}`;

```

```
    const pdfPart = {
        inlineData: {
            data: pdfBuffer.toString('base64'),
            mimeType: 'application/pdf'
        }
    };

    const result = await model.generateContent([prompt, pdfPart]);
    const text = result.response.text();
    const jsonMatch = text.match(/\{[\s\S]*\}/);
    return jsonMatch ? JSON.parse(jsonMatch[0]) : JSON.parse(text);
}
```

3.3.2 Backend: rota de análise

Uma rota `POST /api/proposals/analyze` foi adicionada ao controlador de propostas. Para essa rota, utiliza-se o *middleware* Multer com armazenamento em memória (`memoryStorage`), pois o PDF não precisa ser persistido no S3 durante a análise — apenas processado e descartado:

Listing 2: Rota de análise em `controllers/index.js`.

```
const uploadMemory = multer({
    storage: multer.memoryStorage(),
    fileFilter: (req, file, cb) => {
        if (file.mimetype !== 'application/pdf')
            return cb(new Error('Apenas arquivos PDF sao permitidos.'));
        cb(null, true);
    },
    limits: { fileSize: 10 * 1024 * 1024 }
});

router.post('/analyze',
    verifyJWT,
    checkRole(['STUDENT']),
    uploadMemory.single('pdfFile'),
    async (req, res, next) => {
        try {
            if (!req.file)
                return res.status(400).json({ message: 'PDF obrigatorio
                    .' });
            const feedback = await proposalService.analyzeProposal(req.
                file.buffer);
            res.status(200).json(feedback);
        } catch (error) { next(error); }
    }
);
```

3.3.3 Frontend: fluxo de três etapas

No `StudentDashboard.vue`, o fluxo modal foi reestruturado em três etapas controladas pela variável reativa `modalStep`:

1. **Formulário** (`modalStep = 'form'`): o aluno preenche título, descrição, seleciona o professor e faz upload do PDF;
2. **Análise** (`modalStep = 'analyzing'`): enquanto a requisição à API está em andamento, exibe um *spinner* com mensagem de carregamento;
3. **Feedback** (`modalStep = 'feedback'`): exibe a pontuação (com código de cor: verde ≥ 7, amarelo 4–6, vermelho < 4), o resumo geral e as dicas organizadas por categoria e nível de prioridade. O botão "Confirmar Envio" só aparece nesta etapa.

Esse design garante que o aluno sempre veja o *feedback* da IA antes de confirmar o envio, sem impedir que ele opte por enviar mesmo com pontuação baixa.

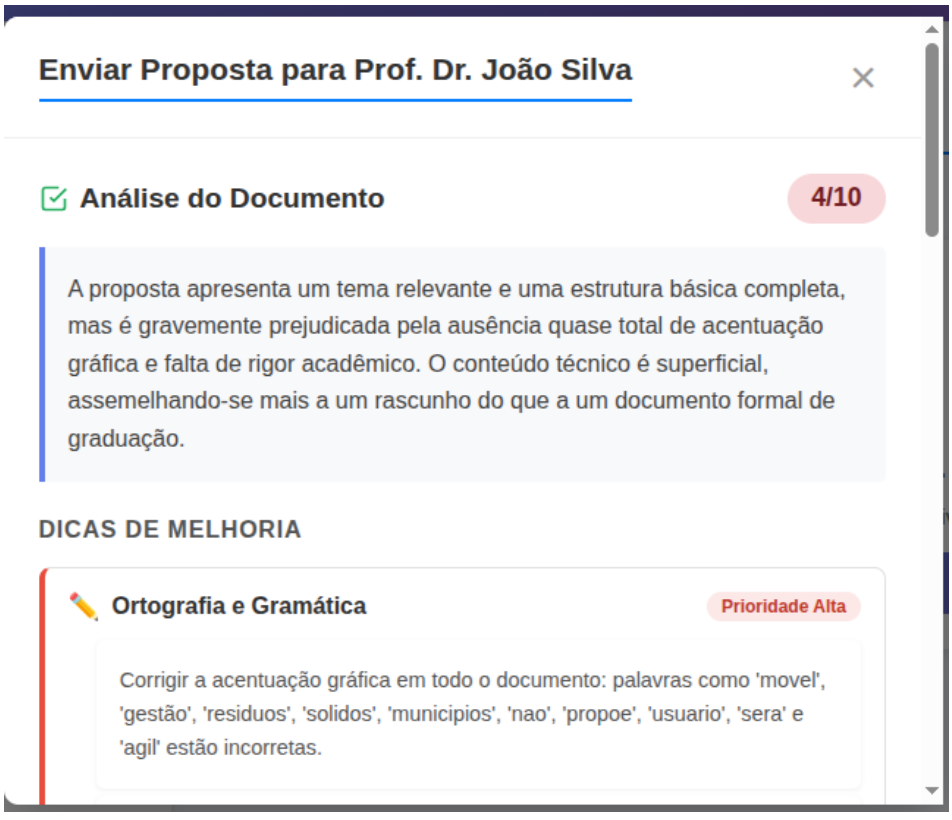

Figura 2: Modal de *feedback* pré-submissão: pontuação com código de cor, resumo geral e dicas organizadas por categoria e nível de prioridade, gerados pelo Gemini antes da confirmação do envio.

3.4 Pipeline Pós-Submissão com AWS Lambda

Após a confirmação do envio pelo aluno, o PDF é carregado para o Amazon S3. Um evento `s3:ObjectCreated:*` aciona automaticamente uma função AWS Lambda que

executa o seguinte fluxo:

1. Recupera o PDF do bucket S3 usando o *key* do objeto;
2. Envia o documento para a API Gemini solicitando um resumo executivo e etiquetas temáticas (*tags*);
3. Chama o endpoint `POST /api/proposals/webhook/ai-metadata` do backend, autenticado por chave de API (`x-api-key`), para persistir os metadados gerados;
4. O painel do professor passa a exibir o resumo e as *tags* de cada proposta recebida.

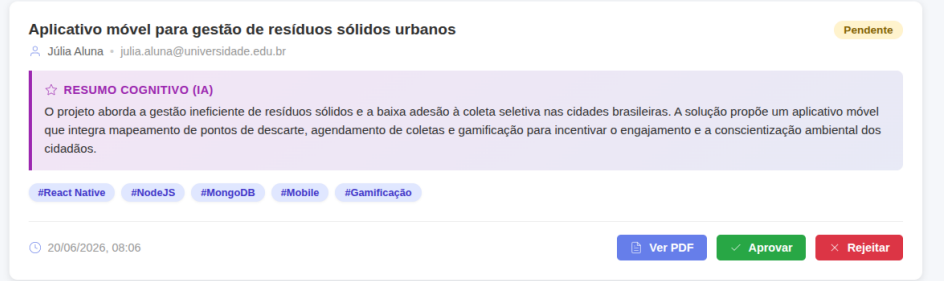

Figura 3: Proposta no painel do professor após o processamento pós-submissão: resumo executivo e etiquetas temáticas gerados automaticamente pela função AWS Lambda via API Gemini.

4 Estudo Experimental: Comparação de Técnicas de Prompt Engineering

4.1 Motivação e Questões de Pesquisa

A proposta do POC II previa um laboratório experimental de *prompt engineering*, conforme descrito na Seção 4 da proposta de trabalho. O objetivo é responder às seguintes questões:

- **Q1**: Qual técnica de *prompting* produz avaliações mais precisas de propostas acadêmicas em português brasileiro?
- **Q2**: Como a qualidade intrínseca do documento afeta o comportamento diferencial entre as três técnicas?
- **Q3**: Existe uma relação entre o número de sugestões geradas e a pontuação atribuída?

4.2 Metodologia Experimental

4.2.1 Corpus de documentos

Foram criados 15 documentos PDF de proposta acadêmica com ReportLab, organizados em três grupos de qualidade predefinida:

- **5 PDFs de alta qualidade (BOM)**: estrutura completa (resumo, palavras-chave, introdução, justificativa, objetivos, hipóteses testáveis, metodologia detalhada com versões de ferramentas, critérios de avaliação quantitativos, cronograma, referências ABNT com publicações recentes), linguagem formal, sem erros ortográficos. Temas: Aprendizado Federado em IoT, Geração de Código com LLM, Otimização de Banco de Dados Grafo, Compressão de Texto Jurídico, Verificação Formal de Contratos Inteligentes;
- **5 PDFs de qualidade intermediária (MÉDIO)**: estrutura parcial (ausência de hipóteses explícitas ou seção de justificativa), metodologia com algum detalhamento, referências básicas, poucos erros de acentuação. Temas: Detecção de Fraude com ML, Diagnóstico de Pneumonia com CNN, Chatbot Universitário, Recomendação de Livros, Análise de Sentimentos;
- **5 PDFs de baixa qualidade (RUIM)**: estrutura informal, linguagem coloquial, múltiplos erros ortográficos, ausência de referências ou metodologia, objetivos vagos. Temas: IA na Saúde (genérico), App de Delivery, Segurança da Informação, Blockchain para Alimentos, Plataforma Educacional.

A Tabela 1 apresenta os identificadores utilizados ao longo da análise para referenciar cada documento.

Tabela 1: Documentos do corpus e respectivos identificadores.

ID	Título	Qualidade
B1	Aprendizado Federado em IoT	BOM
B2	Geração de Código com LLM	BOM
B3	Otimização de Banco de Dados Grafo	BOM
B4	Compressão de Texto Jurídico com NLP	BOM
B5	Verificação Formal de Contratos Inteligentes	BOM
M1	Detecção de Fraude com ML	MÉDIO
M2	Diagnóstico de Pneumonia com CNN	MÉDIO
M3	Chatbot de Suporte Universitário	MÉDIO
M4	Recomendação de Livros	MÉDIO
M5	Análise de Sentimentos em E-commerce	MÉDIO
R1	IA na Saúde (genérico)	RUIM
R2	App de Delivery para Restaurantes	RUIM
R3	Segurança da Informação	RUIM
R4	Blockchain para Rastreio de Alimentos	RUIM
R5	Plataforma Educacional Online	RUIM

4.2.2 Implementação das estratégias

Cada PDF foi analisado pelas três estratégias, totalizando 45 chamadas à API:

Zero-shot Uma única chamada com instrução direta para avaliar o PDF em cinco dimensões (ortografia, clareza, estrutura, qualidade técnica, normas acadêmicas) e retornar o resultado em JSON estruturado.

Few-shot Uma única chamada precedida de três exemplos de avaliações completas (excelente, mediana e fraca) incorporados no *prompt*, orientando o modelo quanto ao tom e ao formato esperado.

Chain-of-Prompts Três chamadas sequenciais com enriquecimento progressivo de contexto:

1. **Extração**: identifica estrutura, erros ortográficos, qualidade de referências e presença de hipóteses;
2. **Classificação**: com base na extração, atribui nota de 1 a 10 a cada dimensão individualmente com justificativa;
3. **Síntese**: com base nas duas etapas anteriores, gera a avaliação final consolidada em JSON, com pontuação ponderada (estrutura e qualidade técnica com peso duplo).

4.2.3 Modelo utilizado

Todos os experimentos foram executados com o modelo `gemini-3-flash-preview`.

4.3 Resultados

4.3.1 Análise por estratégia

Zero-shot Apresentou a maior inconsistência: atribuiu nota 7 ao documento R5 (plataforma educacional informal) e nota 2 ao R1 (IA na saúde, igualmente fraco), sugerindo sensibilidade ao conteúdo temático do documento além de sua qualidade estrutural. Sem exemplos de referência, o modelo tende a ser mais condescendente com entradas de baixa qualidade, o que reduz sua utilidade como instrumento de avaliação discriminativa. O tempo médio de resposta foi de 17,5 s por análise — o intermediário entre as três estratégias.

Few-shot Superou o Zero-shot nos critérios mais relevantes: é mais rápido (média de 14,2 s), atribuiu nota mais alta a documentos de boa qualidade (média 9,0 contra 8,8) e foi mais crítico com documentos fracos (média 3,6 contra 4,6, com maior número de sugestões geradas). Os exemplos calibrados incorporados no *prompt* ancoram o modelo a um padrão de exigência mais elevado, produzindo avaliações mais coerentes com a qualidade real do documento.

Chain-of-Prompts Mostrou-se a estratégia mais **robusta e consistente**: sempre gerou exatamente 10 sugestões por análise, independentemente da qualidade do documento. Identificou problemas que Zero-shot e Few-shot ignoraram — por exemplo, no documento M5 (Sentimentos E-commerce), enquanto Zero-shot atribuiu nota 6 e Few-shot nota 5, Chain-of-Prompts identificou na etapa de extração a ausência de justificativa explícita e seção de palavras-chave, elevando a nota final para 8 com sugestões precisas de correção. O custo é o tempo de processamento (média de 43,6 s por análise, devido às três chamadas sequenciais com pausas de 3 segundos entre elas).

4.4 Discussão

4.4.1 Respostas às questões de pesquisa

Q1 — Qual técnica produz avaliações mais precisas? As três estratégias convergem para documentos de qualidade extrema (BOM ou RUIM muito evidente). A diferença entre elas é mais pronunciada na faixa intermediária, onde documentos com estrutura parcial — presentes no grupo MÉDIO — recebem avaliações bastante distintas dependendo da estratégia utilizada.

Q2 — Como a qualidade do documento afeta o comportamento diferencial? Para documentos claramente excelentes, as três estratégias atribuem notas altas e próximas entre si. Para documentos claramente fracos, também convergem para notas baixas. A maior divergência ocorre em documentos médios e nos extremos atípicos — como R2 (nota 6 no Zero-shot vs. nota 3 no Few-shot) e R5 (nota 7 no Zero-shot vs. nota 3 no Chain-of-Prompts) — sugerindo que cada estratégia captura aspectos distintos da qualidade documental.

Q3 — Relação entre número de sugestões e pontuação? A Figura 4 evidencia que Zero-shot e Few-shot apresentam correlação inversa entre nota e número de sugestões (documentos RUIM geram mais sugestões), enquanto o Chain-of-Prompts normaliza a saída em 10 sugestões independentemente da qualidade.

4.4.2 Análise do número de sugestões por documento

A Figura 4 sumariza as três métricas coletadas — pontuação média, número de sugestões geradas e tempo de execução — para as três estratégias e grupos de qualidade.

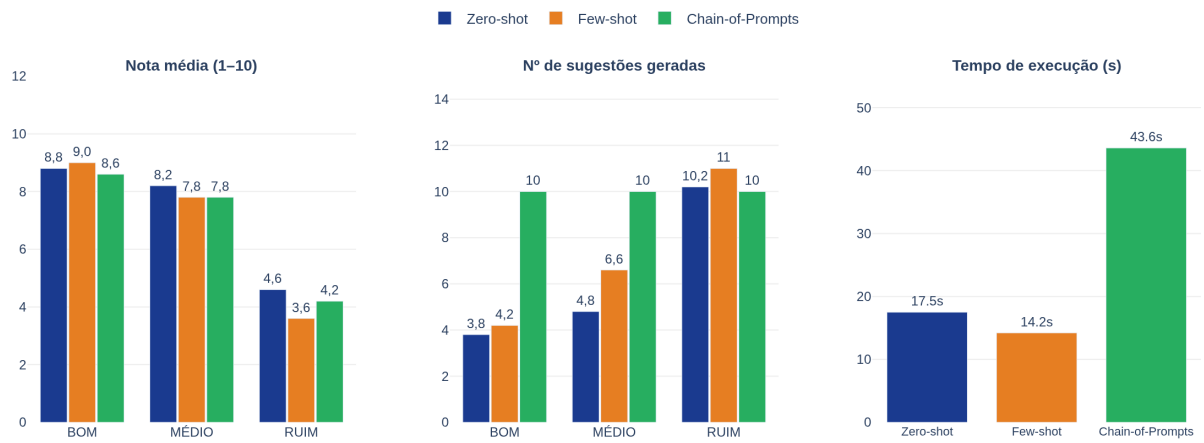

Figura 4: Comparação entre as três estratégias de *prompting* nas três métricas coletadas. Da esquerda para a direita: nota média (1–10), número de sugestões geradas e tempo de execução por chamada à API.

Três padrões se destacam:

Zero-shot: há uma forte correlação inversa entre pontuação e número de sugestões — documentos BOM recebem consistentemente 3 a 5 sugestões, enquanto documentos RUIM chegam a 14. O padrão quebra em dois casos atípicos: M5 recebeu 10 sugestões apesar de nota 6 (quase na faixa de RUIM), e R5 recebeu apenas 3 sugestões com nota 7, sugerindo que o modelo tratou o conteúdo temático desse documento (plataforma educacional) como mais adequado do que sua estrutura indicaria. Esses outliers revelam que o Zero-shot é sensível ao tema além da forma.

Few-shot: a correlação entre sugestões e qualidade é presente, mas com maior variância. R4 recebeu 16 sugestões — o máximo absoluto do experimento — enquanto B3

recebeu apenas 2. O caso de R4 indica que os exemplos incorporados no *prompt* amplificam a percepção de problemas em documentos fracos com múltiplos vícios estruturais. M5 e R1 receberam ambos 10 sugestões, mas com notas muito distintas (5 e 2, respectivamente), sugerindo que o Few-shot usa sugestões adicionais de forma menos discriminativa do que o Zero-shot.

Chain-of-Prompts: produziu exatamente 10 sugestões para 14 dos 15 documentos, com exceção de R2 (11 sugestões). Essa normalização é um efeito direto da etapa de síntese, que consolida e padroniza a saída independentemente do volume de problemas identificados nas etapas anteriores. A nota varia (de 3 a 9), mas o número de sugestões permanece estável, indicando que a estratégia separa explicitamente a avaliação quantitativa (nota) da produção de conteúdo acionável (sugestões).

4.4.3 Pontos fortes e fracos de cada estratégia

Zero-shot O principal ponto forte do Zero-shot é a menor latência do experimento (média de 17,5 s) combinada à implementação mais simples — um único *prompt* e uma única chamada à API — o que facilita manutenção e ajustes futuros. O ponto fraco central é a variância elevada: documentos estruturalmente semelhantes podem receber números muito diferentes de sugestões (3 vs. 14 no grupo RUIM), e a sensibilidade temática observada — R5 com nota 7 apesar de qualidade estrutural baixa — compromete a confiabilidade da nota como indicador de qualidade. Sem exemplos de referência no *prompt*, o modelo tende a condescender com entradas de baixa qualidade, reduzindo sua utilidade como instrumento de avaliação discriminativa.

Few-shot Os exemplos incorporados no *prompt* calibram o modelo a um padrão de exigência mais elevado, produzindo a maior nota média no grupo BOM (9,0) e a menor no grupo RUIM (3,6), e a menor latência geral (14,2 s). Esses atributos tornam o Few-shot a estratégia com melhor relação custo-benefício para o contexto deste trabalho. O ponto fraco é a amplificação de defeitos em documentos muito fracos — R4 recebeu 16 sugestões, o máximo absoluto do experimento — o que pode tornar a saída excessivamente extensa. Além disso, qualquer evolução no formato das propostas do DCC exigiria atualizar manualmente os exemplos no *prompt*.

Chain-of-Prompts A decomposição em três etapas sequenciais (extração, classificação e síntese) produz as avaliações mais fundamentadas do experimento: no caso de M5, o modelo identificou na etapa de extração ausências estruturais que os outros métodos ignoraram, gerando a maior nota entre as estratégias para esse documento (8 vs. 6 e 5). A estabilidade da saída — exatamente 10 sugestões em praticamente todos os documentos — garante consistência de formato e viabiliza o log intermediário de cada fase para auditoria do processo. As desvantagens são o tempo de processamento elevado (43,6 s em média,

devido às três chamadas com pausas entre elas) e a normalização do volume de sugestões, que elimina esse indicador como sinal de qualidade — documentos BOM e RUIM recebem igualmente 10 sugestões, embora de naturezas distintas. A orquestração de chamadas sequenciais também introduz complexidade operacional: uma falha na segunda ou terceira chamada, após tokens já consumidos na anterior, exige mecanismos de retentativa por etapa.

4.5 Implicações para o sistema socket.cc

O conjunto de resultados permite estabelecer uma **hierarquia parcial** entre as estratégias. O **Few-shot supera o Zero-shot** nos critérios mais relevantes: é mais rápido (14,2 s contra 17,5 s), mais preciso com documentos de boa qualidade (nota 9,0 contra 8,8) e mais crítico com documentos fracos (nota 3,6 contra 4,6). O Zero-shot, sem exemplos de referência no *prompt*, tende a condescender com entradas de baixa qualidade — justamente o cenário em que a avaliação discriminativa é mais necessária.

A comparação entre **Few-shot e Chain-of-Prompts**, por sua vez, configura um *trade-off* genúino. O Few-shot é aproximadamente três vezes mais rápido (14,2 s contra 43,6 s) e preserva o número de sugestões como sinal diagnóstico da qualidade do documento — documentos fracos recebem mais sugestões, o que carrega informação sobre a qualidade da entrada. O Chain-of-Prompts, ao decompor o problema em três etapas sequenciais (extração, classificação e síntese), tende a produzir raciocínio mais estruturado e saída em formato JSON mais confiável; a consistência de exatamente 10 sugestões para todos os documentos, porém, elimina esse sinal diagnóstico. O Few-shot é mais volátil na forma, mas mais expressivo no conteúdo; o Chain-of-Prompts é mais previsível na estrutura, mas menos sensível à variação de qualidade entre documentos. A escolha entre as duas estratégias depende, portanto, das prioridades do contexto de aplicação.

5 Uso de Inteligência Artificial no Desenvolvimento

O assistente de código foi utilizado como ferramenta de auxílio durante a implementação do sistema. O os trechos de código gerado foram revisados e integrados manualmente pelo autor.

6 Conclusões e Trabalhos Futuros

6.1 Conclusões

Este trabalho apresentou a evolução do sistema *socket.cc* de um repositório passivo de documentos para uma plataforma de inteligência documental. As principais contribuições

foram:

1. **Módulo de análise pré-submissão**: a integração da API Google Gemini com o fluxo de submissão do frontend, através de um modal em três etapas, permite ao aluno receber *feedback* estruturado antes de enviar sua proposta ao professor. Esse recurso tem potencial de reduzir rejeições por problemas formais e melhorar a qualidade média das propostas recebidas;

2. **Pipeline serverless pós-submissão**: o uso de AWS Lambda em resposta a eventos S3 viabilizou o processamento assíncrono de documentos sem impactar a experiência do usuário e sem custo contínuo de infraestrutura;

3. **Estudo de prompt engineering**: o experimento com 15 documentos e 45 chamadas à API estabeleceu uma hierarquia parcial entre as técnicas. O Few-shot supera o Zero-shot nos critérios mais relevantes — velocidade, precisão com documentos de boa qualidade e rigor com documentos fracos. A comparação entre Few-shot e Chain-of-Prompts revela um *trade-off* genúino: o primeiro é mais rápido (14,2 s) e preserva o número de sugestões como sinal diagnóstico; o segundo é mais consistente e estruturado na saída, ao custo de maior latência (43,6 s) e perda desse sinal diagnóstico;

4. **Infraestrutura de produção**: a implantação em EC2 demonstrou a viabilidade de hospedar o sistema em infraestrutura de baixo custo (Free Tier AWS) com estabilidade aceitável para uso acadêmico.

Do ponto de vista do conhecimento adquirido, o projeto exercitou competências em desenvolvimento full-stack (Vue.js, Node.js, Prisma), computação em nuvem (AWS EC2, S3, Lambda, IAM), integração de LLMs via API e design de experimentos controlados para avaliação de técnicas de IA.

6.2 Trabalhos Futuros

- **Componentes inteligentes no painel do professor**: exibição do resumo executivo e das *tags* geradas pela pipeline Lambda diretamente na interface de triagem, com filtros por tecnologia;

- **Validação com propostas reais**: aplicar as três estratégias de *prompting* a propostas reais de alunos do DCC/UFMG, com avaliação humana por professores para medir a correlação entre as notas do modelo e o julgamento especializado;

- **Fine-tuning**: explorar o ajuste fino de um modelo open-source (ex.: LLaMA 3 ou Mistral) em um corpus de propostas avaliadas do departamento, reduzindo a dependência de APIs externas;

- **Autenticação institucional**: integrar SSO com as credenciais UFMG para eliminar o cadastro manual de usuários.

Referências

[1] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., *et al. Language Models are Few-Shot Learners*. Advances in Neural Information Processing Systems, v. 33, p. 1877–1901, 2020.

[2] Google DeepMind. *Gemini: A Family of Highly Capable Multimodal Models*. Technical Report, Google DeepMind, 2023. Disponível em: `https://arxiv.org/abs/2312.11805`

[3] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., *et al. A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT*. arXiv preprint arXiv:2302.11382, 2023.

[4] Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., Iwasawa, Y. *Large Language Models are Zero-Shot Reasoners*. In: *Advances in Neural Information Processing Systems*, v. 35, 2022.

[5] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., *et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. In: *Advances in Neural Information Processing Systems*, v. 35, 2022.

[6] Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C.-C., Khandelwal, A., Pu, Q., *et al. Cloud Programming Simplified: A Berkeley View on Serverless Computing*. arXiv preprint arXiv:1902.03383, 2019.

[7] Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., Su, Z. ArnetMiner: Extraction and Mining of Academic Social Networks. In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 990–998, 2008.

[8] Li, J., Tang, T., Zhao, W. X., Nie, J.-Y., Wen, J.-R. *Pretrained Language Models for Text Generation: A Survey*. arXiv preprint arXiv:2201.05273, 2022.

[9] Amazon Web Services. *AWS Lambda Developer Guide*, 2024. Disponível em: `https://docs.aws.amazon.com/lambda/latest/dg/welcome.html`

[10] Amazon Web Services. *Amazon S3 User Guide*, 2024. Disponível em: `https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html`

[11] Prisma Data, Inc. *Prisma ORM Documentation*, 2024. Disponível em: `https://www.prisma.io/docs`

[12] Vue.js Core Team. *Vue.js 3 Documentation*, 2024. Disponível em: `https://vuejs.org/guide/introduction.html`