Sistema Estruturado e Híbrido de Recomendação de Problemas de Programação no Codeforces: Perfil Multitag e Recuperação Semântica

Júlia Akemi Koba

Universidade Federal de Uberlândia
Faculdade de Computação
Bacharelado em Sistemas de Informação

Monte Carmelo - MG
2026

Júlia Akemi Koba

Sistema Estruturado e Híbrido de Recomendação de Problemas de Programação no Codeforces: Perfil Multitag e Recuperação Semântica

Trabalho de Conclusão de Curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Área de concentração: Sistemas de Informação

Orientador: Professor Dr. Carlos Cesar Mansur Tuma

Monte Carmelo - MG
2026

Este trabalho é dedicado à minha avó e meu avô, que sempre me inspiraram e me apoiaram em todos os momentos.

Agradecimentos

Agradeço, em primeiro lugar, aos meus pais, pelo apoio incondicional, pelo incentivo e pela dedicação presentes em cada etapa da minha jornada. Obrigada por acreditarem em mim, por respeitarem minhas escolhas e por me ajudarem a continuar mesmo nos momentos de cansaço e incerteza.

Ao meu irmão, agradeço pela amizade, pelo companheirismo e pela presença constante.

Aos meus avós, pelo amor, pelos ensinamentos e pela inspiração que levarei comigo durante toda a vida.

Agradeço especialmente à professora Daniele, que idealizou este projeto e teve uma participação muito importante no início de seu desenvolvimento. Sou muito grata por todo o apoio, carinho, atenção e dedicação que ela demonstrou durante o período em que esteve presente. Suas orientações e sua confiança foram fundamentais para que esta pesquisa começasse a tomar forma. Mesmo não acompanhando as etapas finais deste trabalho, sua contribuição permanece presente na origem da proposta e em tudo o que foi desenvolvido a partir dela.

Ao professor Carlos Tuma, meu atual orientador, agradeço pela orientação, pela paciência e pelo cuidado durante a continuidade e a conclusão deste trabalho. Suas observações, questionamentos e sugestões foram essenciais para melhorar a pesquisa, esclarecer suas contribuições e tornar o texto mais consistente. Agradeço também pela disponibilidade em acompanhar as mudanças realizadas e por me ajudar a compreender melhor as decisões tomadas ao longo do projeto.

Agradeço a todos os professores que participaram da minha formação acadêmica e contribuíram para meu aprendizado durante a graduação. Cada disciplina, orientação e experiência teve importância na construção do conhecimento necessário para chegar até aqui.

Por fim, agradeço aos meus amigos próximos, que estiveram ao meu lado oferecendo apoio, compreensão e motivação nos momentos mais desafiadores. A presença de cada um tornou essa trajetória mais leve e significativa.

"A inteligência é um presente, mas o coração é mais humano do que qualquer máquina pode ser."

(Keyes, Flores para Algernon)

Resumo

As plataformas de Juízes Online oferecem grandes catálogos de problemas de programação, mas a escolha de atividades compatíveis com a experiência de cada competidor pode ser difícil, principalmente porque o nível pode variar entre diferentes assuntos. Este trabalho teve como objetivo desenvolver e avaliar um sistema híbrido de recomendação de problemas do Codeforces baseado em perfil de proficiência por tags, filtros estruturados, recuperação semântica por busca vetorial baseada em embeddings e seleção opcional por um Modelo de Linguagem de Grande Escala (LLM). A avaliação utilizou registros públicos de 85 competidores, cujos históricos de problemas resolvidos foram divididos cronologicamente entre o passado observado e o futuro observado. O passado observado foi utilizado para selecionar as três tags mais frequentes, calcular o nível operacional de cada uma e produzir as recomendações. Foram comparadas quatro abordagens: estruturada sem LLM, estruturada com LLM, híbrida sem LLM e híbrida com LLM. No total, foram avaliadas 340 listas e 3.400 recomendações. As abordagens híbridas alcançaram Hit Rate@10 de 44,71%, enquanto as abordagens estruturadas obtiveram 9,41% e 10,59%. Na recuperação híbrida, a utilização da LLM manteve o mesmo Hit Rate@10 e ampliou a diversidade semântica média de 0,2121 para 0,2169, além de aumentar a variedade global de 207 para 221 problemas distintos. Todas as recomendações permaneceram dentro da faixa de dificuldade definida para cada tag. Os resultados apoiaram as três hipóteses no cenário avaliado: o procedimento híbrido apresentou maior correspondência com o futuro observado, a curadoria por LLM ampliou a diversidade e a variedade das listas, e os filtros preservaram o controle da dificuldade.

Palavras-chave: Sistemas de Recomendação, Programação Competitiva, Codeforces, RAG, Recuperação Semântica.

Abstract

Online Judge platforms provide extensive catalogs of programming problems, but selecting activities suited to each competitor's experience can be challenging, particularly because proficiency may vary across topics. This study aimed to develop and evaluate a hybrid Codeforces problem recommender system based on tag-specific proficiency profiles, structured filters, semantic retrieval through embedding-based vector search, and optional curation using a Large Language Model (LLM). The evaluation used public records from 85 competitors, whose solved-problem histories were chronologically divided into an observed past and an observed future. The observed past was used to identify the three most frequent tags, calculate the operational proficiency level for each tag, and generate recommendations. Four approaches were compared: structured retrieval without an LLM, structured retrieval with an LLM, hybrid retrieval without an LLM, and hybrid retrieval with an LLM. In total, 340 recommendation lists comprising 3,400 recommendations were evaluated. The hybrid approaches achieved a Hit Rate@10 of 44.71%, whereas the structured approaches achieved 9.41% and 10.59%. In the hybrid setting, the use of the LLM preserved the same Hit Rate@10, increased mean semantic diversity from 0.2121 to 0.2169, and expanded the overall variety from 207 to 221 distinct problems. All recommendations remained within the difficulty range defined for each tag. The results supported all three research hypotheses in the evaluated scenario: the hybrid procedure produced greater correspondence with the observed future, LLM-based curation increased the diversity and variety of the recommendation lists, and the structured filters preserved difficulty control.

Keywords: Recommender Systems, Competitive Programming, Codeforces, Retrieval-Augmented Generation, Semantic Retrieval.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API Interface de Programação de Aplicações - *Application Programming Interface*

CSV Valores Separados por Vírgulas - *Comma-Separated Values*

CTF *Capture The Flag*

HNSW Mundo Pequeno Navegável Hierárquico - *Hierarchical Navigable Small World*

ILD Diversidade Intra-Lista - *Intra-List Diversity*

JSON Notação de Objetos JavaScript - *JavaScript Object Notation*

LLM Modelo de Linguagem de Grande Escala - *Large Language Model*

MAE Erro Absoluto Médio - *Mean Absolute Error*

OJ Juiz Online - *Online Judge*

RAG Geração Aumentada por Recuperação - *Retrieval-Augmented Generation*

SMAS Modelo de Múltiplas Habilidades e Capacidades dos Estudantes - *Students' Multiple Abilities and Skills Model*

TF-IDF Frequência do Termo–Inverso da Frequência nos Documentos - *Term Frequency–Inverse Document Frequency*

Sumário

1 Introdução

A programação competitiva consolida-se atualmente não apenas como um método de resolução de problemas sob rigorosas restrições de tempo e recursos computacionais, mas como um paradigma educacional focado no domínio prático de habilidades (LIM, 2026). Segundo o autor, essa prática fundamentada no aprendizado competitivo não está reservada exclusivamente ao desenvolvimento de software tradicional, estendendo-se a diversas áreas tecnológicas e formatos de disputa, como os *hackathons* e as competições de segurança da informação, a exemplo dos eventos *Capture The Flag* (CTF). No contexto de treinamento de algoritmos, plataformas de Juízes Online (OJs), como o Codeforces, disponibilizam grandes catálogos de problemas e avaliam automaticamente as soluções submetidas (TOLEDO; MOTA; MARTÍNEZ, 2018; YOSHIMURA et al., 2022; LEE; LAO, 2019).

Embora esse volume de conteúdo ofereça muitas possibilidades de prática, ele também dificulta a escolha do próximo problema, principalmente para competidores iniciantes. Além disso, o rating geral é uma pontuação numérica que estima a habilidade do competidor com base em seu histórico de desempenho e não representa necessariamente a experiência em cada assunto isolado. Um competidor pode resolver problemas de determinada dificuldade em matemática e apresentar um histórico diferente em grafos ou manipulação de textos (ZAFFALON et al., 2022; RAMESH; KANNIMUTHU, 2023).

Nesse contexto, este trabalho investiga a recomendação de problemas do Codeforces por meio de um perfil separado por *tags*. A proposta combina regras de assunto e dificuldade com recuperação semântica por busca vetorial baseada em *embeddings* e seleção opcional por uma Modelo de Linguagem de Grande Escala (LLM). Dessa forma, o sistema procura selecionar problemas ainda não resolvidos, compatíveis com o nível operacional calculado para cada assunto.

1.1 Motivação

Os OJs permitem que o competidor pratique programação e receba retorno automático sobre seu código. Entretanto, essas plataformas normalmente oferecem pouca orientação individual sobre qual problema deve ser resolvido em seguida. A existência de filtros por *tag* e dificuldade ajuda na navegação, mas ainda exige que o próprio competidor conheça seu nível em cada assunto (TOLEDO; MOTA; MARTíNEZ, 2018; LEE; LAO, 2019; YOSHIMURA et al., 2022).

Sistemas de recomendação podem reduzir essa sobrecarga ao selecionar uma quantidade menor de problemas com base no histórico observado. No contexto educacional, essa seleção pode organizar a prática e considerar diferenças entre os assuntos já exercitados

pelo competidor (RICCI; ROKACH; SHAPIRA, 2015; JULIANTI et al., 2022; MUEPU; WATANOBE; AMIN, 2025).

O Construcionismo, a Zona de Desenvolvimento Proximal e a Avaliação Silenciosa oferecem apoio conceitual para interpretar a prática, a escolha da dificuldade e o uso do histórico na organização das recomendações (PAPERT, 1980; VYGOTSKY, 1978; SHUTE, 2011).

Outra motivação desta pesquisa reside na viabilidade de aplicar a busca vetorial, orientada por representações semânticas (*embeddings*), para refinar e reordenar a seleção de problemas que já pertencem a uma mesma categoria temática e a uma mesma faixa de dificuldade. Com isso, a seleção pode considerar uma estimativa de proximidade semântica depois da aplicação dos filtros estruturados (LEWIS et al., 2020; MUEPU; WATANOBE; AMIN, 2025).

A inclusão da busca vetorial baseada em *embeddings* e de um LLM acrescenta novas etapas ao fluxo do sistema. Para observar o efeito de cada componente, esta pesquisa compara abordagens com e sem busca vetorial baseada em *embeddings* e com e sem LLM.

1.2 Problema de Pesquisa

O problema abordado nesta pesquisa é a seleção personalizada de problemas em um catálogo amplo de programação competitiva. O sistema precisa considerar que um mesmo usuário pode possuir proficiências diferentes em cada *tag*, evitar problemas já resolvidos e manter as recomendações em uma faixa de dificuldade definida.

A partir desse problema, foi formulada a seguinte pergunta de pesquisa:

> Em que medida um procedimento híbrido, que combina filtros por *tag* e dificuldade com recuperação semântica por busca vetorial baseada em *embeddings*, melhora a correspondência das recomendações com o conjunto de avaliação (futuro observado) de usuários do Codeforces com *rating* entre 600 e 1400, e qual é o efeito adicional da LLM sobre a diversidade?

1.3 Hipóteses

- **H1**: o procedimento híbrido apresenta maior correspondência com o conjunto de avaliação (futuro observado) do que a recuperação exclusivamente estruturada;
- **H2**: a seleção realizada pela LLM aumenta a diversidade das recomendações em relação às abordagens equivalentes sem LLM;
- **H3**: os filtros mantêm os problemas recomendados dentro da faixa de dificuldade calculada para cada *tag*, mesmo com a utilização da busca vetorial baseada em *embeddings* e da LLM.

Essas hipóteses tratam de correspondência *offline*, diversidade e adequação operacional.

1.4 Objetivos

1.4.1 Objetivo Geral

Desenvolver e avaliar um sistema híbrido de recomendação de problemas do Codeforces que utilize um perfil de proficiência por *tags*, filtros estruturados, recuperação semântica por busca vetorial baseada em *embeddings* e seleção opcional por LLM.

1.4.2 Objetivos Específicos

- ❏ coletar e organizar informações públicas de usuários, submissões e problemas por meio da Interface de Programação de Aplicações (API) do Codeforces;
- ❏ construir um perfil separado para cada *tag*, considerando a quantidade e a dificuldade dos problemas resolvidos;
- ❏ ordenar as *tags* da menor para a maior proficiência para orientar a distribuição das recomendações;
- ❏ indexar os problemas por meio de *embeddings* e armazená-los em um banco vetorial;
- ❏ combinar filtros estruturados com recuperação semântica por busca vetorial baseada em *embeddings*;
- ❏ utilizar uma LLM somente sobre os problemas previamente recuperados e impedir a inclusão de identificadores que não estejam nesse conjunto;
- ❏ comparar as abordagens estruturada sem LLM, estruturada com LLM, híbrida sem LLM e híbrida com LLM;
- ❏ avaliar as recomendações por meio de *Precisão@10*, *Hit Rate@10* e erro absoluto médio de *rating*;
- ❏ analisar de forma complementar o indicador de avanço por *tag* e a diversidade semântica das listas.

1.5 Contribuições

A primeira contribuição é a modelagem de um perfil multitag do usuário. Em vez de utilizar uma nota geral, o nível do competidor é calculado de forma independente para

cada assunto. As *tags* são então ordenadas da menor para a maior proficiência, o que permite direcionar as recomendações para as maiores deficiências técnicas do estudante.

A segunda contribuição é o desenvolvimento de um *pipeline* híbrido de recuperação de informação. Essa arquitetura integra regras rígidas de negócio, como filtros de dificuldade e histórico, com uma busca semântica baseada em representações vetoriais (*embeddings*). Esse *pipeline* garante que a comparação semântica ocorra apenas entre problemas elegíveis e adequados ao nível do competidor.

A terceira contribuição é a organização de um fluxo de Geração Aumentada por Recuperação (RAG) em que a LLM atua estritamente como um agente curador. A validação programática dos identificadores devolvidos pela LLM impede alucinações e garante que o modelo não invente problemas fora do catálogo recuperado.

A quarta contribuição é a avaliação pareada e temporal das abordagens. O protocolo de avaliação isolou o passado e o futuro observado dos mesmos usuários. Isso permitiu quantificar de forma exata o impacto isolado da busca vetorial e da LLM sobre as métricas de acerto e diversidade.

1.6 Organização da Monografia

Além desta introdução, o trabalho está organizado em mais três capítulos. O Capítulo 2 apresenta a Fundamentação Teórica e os Trabalhos Relacionados, incluindo programação competitiva, Juízes Online, teorias educacionais, sistemas de recomendação, *embeddings*, recuperação híbrida, Inteligência Artificial Generativa, RAG, métricas de avaliação e a comparação com pesquisas anteriores.

O Capítulo 3 reúne o método de avaliação, a preparação dos dados, a construção do perfil multitag, a indexação dos *embeddings* no banco vetorial, as quatro abordagens, os experimentos e a análise dos resultados. Por fim, o Capítulo 4 apresenta as conclusões e as possibilidades de trabalhos futuros.

2 Fundamentação Teórica

Este capítulo apresenta os conceitos necessários para compreender sistemas de recomendação de problemas de programação e discute trabalhos relacionados a esse tema. Primeiro, são abordados a programação competitiva, os OJs e o Codeforces. Em seguida, são apresentadas teorias educacionais, conceitos de sistemas de recomendação, recuperação de informação, LLM, RAG e métricas de avaliação. Ao final, os trabalhos relacionados são descritos e comparados com a proposta desenvolvida.

2.1 Programação Competitiva e Juízes Online

A programação competitiva e os OJs, são conceitos importantes para o ensino e a prática de programação, principalmente no desenvolvimento de habilidades de resolução de problemas e pensamento computacional (MUEPU; WATANOBE; AMIN, 2025; YOSHIMURA et al., 2022; LEE; LAO, 2019). No contexto desta pesquisa será relacionada com maratonas de programação.

2.1.1 Juízes Online

OJs são aplicações que mantêm coleções de problemas de programação e automatizam a avaliação das soluções enviadas pelos usuários (TOLEDO; MOTA; MARTíNEZ, 2018; YOSHIMURA et al., 2022; LEE; LAO, 2019). Sua principal função é fornecer retorno automático sobre a correção de um código-fonte. Uma submissão pode receber resultados como "Aceita" (*Accepted*), "Resposta Errada" (*Wrong Answer*), "Limite de Tempo Excedido" (*Time Limit Exceeded*) ou "Erro de Execução" (*Runtime Error*), entre outros. Esse retorno permite que o usuário analise sua solução, faça alterações e realize novas tentativas (TOLEDO; MOTA; MARTíNEZ, 2018).

2.1.2 Programação Competitiva

É uma atividade intelectual que consiste em resolver problemas algorítmicos complexos sob restrições de tempo e recursos computacionais (YOSHIMURA et al., 2022). Os participantes, além de criar uma solução correta, têm que garantir que ela seja eficiente. Os OJs são o ambiente padrão para isso, já que fornecem problemas e avaliam as submissões de forma imparcial e padronizada (TOLEDO; MOTA; MARTíNEZ, 2018). O ambiente dos OJs muitas vezes inclui ranking, o que motiva os usuários a aprimorarem suas habilidades (TOLEDO; MOTA; MARTíNEZ, 2018).

Um problema de programação pode envolver diferentes conhecimentos e permitir mais de uma estratégia de solução. Zaffalon et al. (2022) destacam que as habilidades do estudante não devem ser tratadas necessariamente como uma capacidade única. Um usuário pode apresentar maior experiência em matemática, por exemplo, e menor experiência em grafos.

2.1.3 Histórico, Evolução e Exemplos

Embora a utilização de sistemas de avaliação automática para tarefas de programação seja uma prática já amplamente consolidada no meio acadêmico, ela obteve aprimoramentos significativos e alcance global com a popularização da internet. As primeiras referências a OJs na literatura citam como ferramentas úteis para a avaliação de códigos-fonte que solucionam problemas propostos (TOLEDO; MOTA; MARTíNEZ, 2018). Plataformas como *University of Valladolid Online Judge* (UVaOJ) e o *Peking University Online Judge* acumularam centenas de milhares de usuários e milhares de problemas e isso foi essencial para a popularização da prática (TOLEDO; MOTA; MARTíNEZ, 2018).

Atualmente, o cenário é composto por plataformas que atraem uma comunidade global, como Codeforces e TopCoder, frequentemente citados como ambientes padrão de treinamento (LEE; LAO, 2019). O sucesso e a alta adoção dos OJs levaram a um enorme crescimento no número de problemas disponíveis (TOLEDO; MOTA; MARTíNEZ, 2018). Esse grande volume de conteúdo criou um desafio maior: a sobrecarga de informação, que torna difícil para os usuários, principalmente para os iniciantes, a seleção de problemas adequados ao seu nível de conhecimento (TOLEDO; MOTA; MARTíNEZ, 2018) (LEE; LAO, 2019). A escolha de um problema muito acima de seu nível pode levar à frustração e ao desengajamento (YOSHIMURA et al., 2022) (MUEPU; WATANOBE; AMIN, 2025).

2.1.4 Codeforces

O Codeforces é uma plataforma de programação competitiva que funciona de forma parecida aos juízes online. Os usuários participam de competições, resolvem problemas e enviam suas soluções para avaliação automática. A plataforma também pode ser utilizada para prática individual, pois mantém disponível um catálogo de problemas publicados em competições anteriores (LEE; LAO, 2019) (Codeforces, 2026).

No ecossistema do Codeforces, o termo *rating* é fundamental e possui dupla aplicação: ele serve para classificar tanto os usuários quanto os problemas. O *rating* do usuário (ver Figura 1) é uma pontuação numérica baseada no sistema Elo, que é um método matemático, originalmente criado para o xadrez, que calcula a proficiência relativa dos competidores com base no resultado histórico de suas interações em torneios. Esse valor flutua após cada competição oficial, refletindo a habilidade geral do usuário.

Em contrapartida, o *rating* do problema (ver Figura 2) indica o nível de dificuldade estimado daquela questão. Ele é calculado a partir da proporção de usuários com determinados *ratings* que conseguiram resolvê-la durante as competições. Dessa forma, um usuário pode possuir um *rating* geral mediano, mas ser capaz de resolver problemas com *ratings* (dificuldades) muito superiores em assuntos específicos nos quais tem mais experiência (CODEFORCES, 2026a; 2026b).

Segue abaixo a diferenciação dos *ratings*:

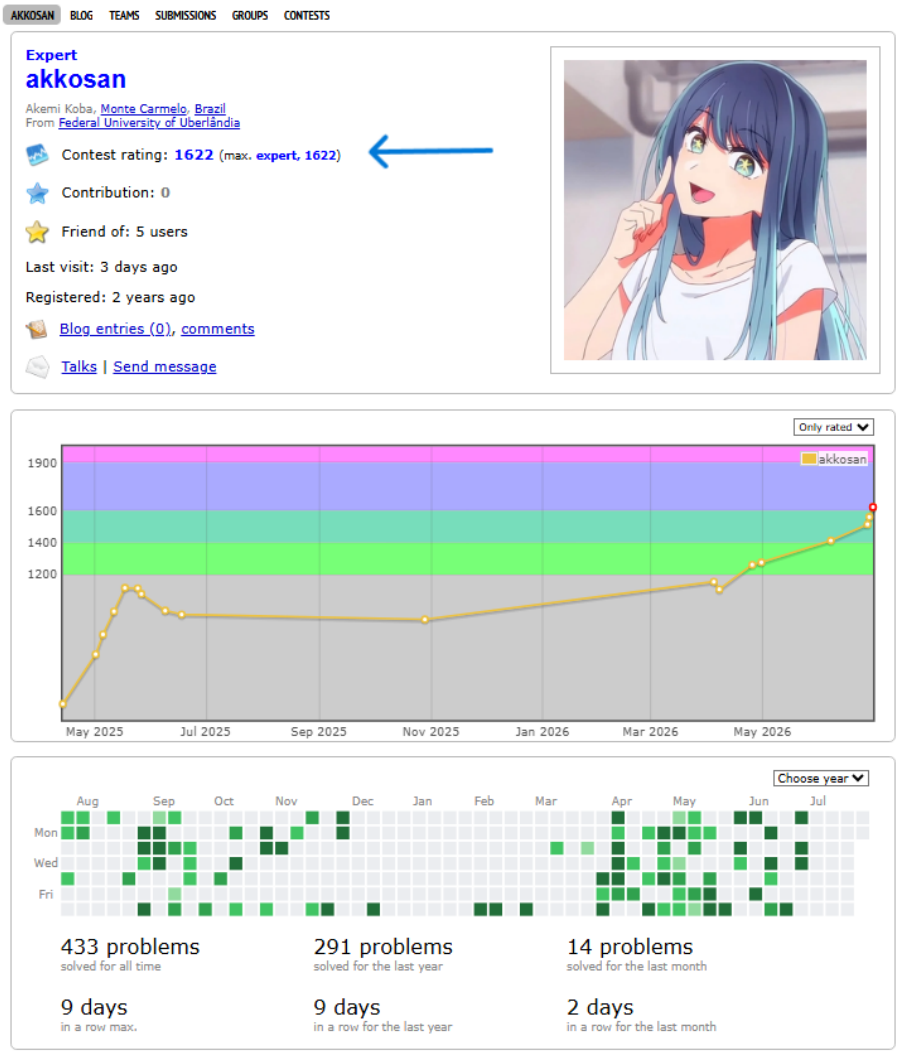

Figura 1 – *Rating* do usuário - Fonte: (Codeforces, 2026)

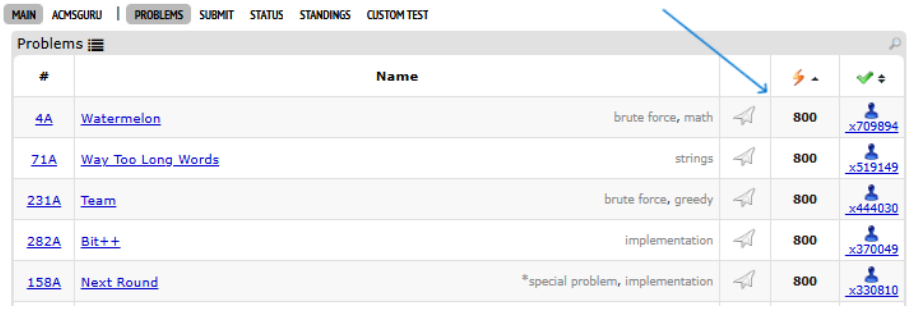

Figura 2 – *Rating* do problema - Fonte: (Codeforces, 2026)

Os problemas também podem possuir *tags*, como math, graphs, strings, dynamic programming. Essas *tags* mostram os principais assuntos relacionados à solução. No entanto, um mesmo problema pode possuir várias *tags*, pois pode precisar mais de um conhecimento.

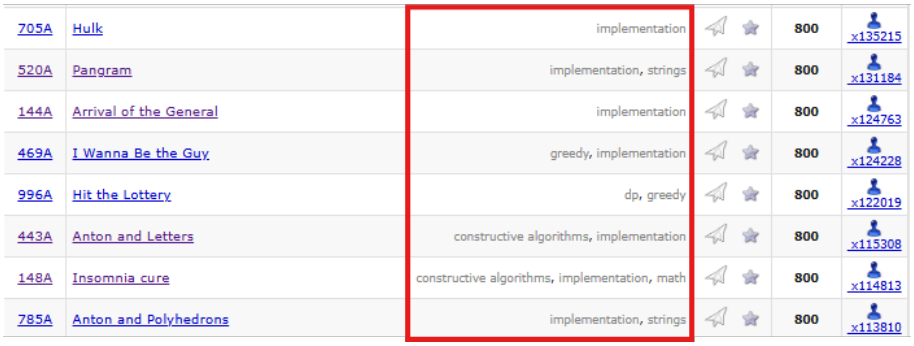

Figura 3 – *Tags* dos problemas - Fonte: (Codeforces, 2026)

O Codeforces também oferece uma API pública que permite consultar dados da plataforma em formato Notação de Objetos JavaScript (JSON), incluindo informações sobre usuários, problemas e submissões (Codeforces, 2026). Os métodos utilizados para coletar esses dados serão apresentados no capítulo de metodologia e desenvolvimento do sistema.

2.1.5 Kaggle

O Kaggle (ver Figura 4) é uma plataforma fundada em 2010 e adquirida pela Google LLC em 2017. É reconhecida como uma das maiores plataformas colaborativas on-line voltadas às áreas de ciência de dados e aprendizado de máquina. O ambiente atua tanto como repositório de conjuntos de dados (*datasets*), públicos e privados, quanto como espaço para competições de modelagem preditiva, discussões e compartilhamento de códigos executados em nuvem (Kaggle, 2026)

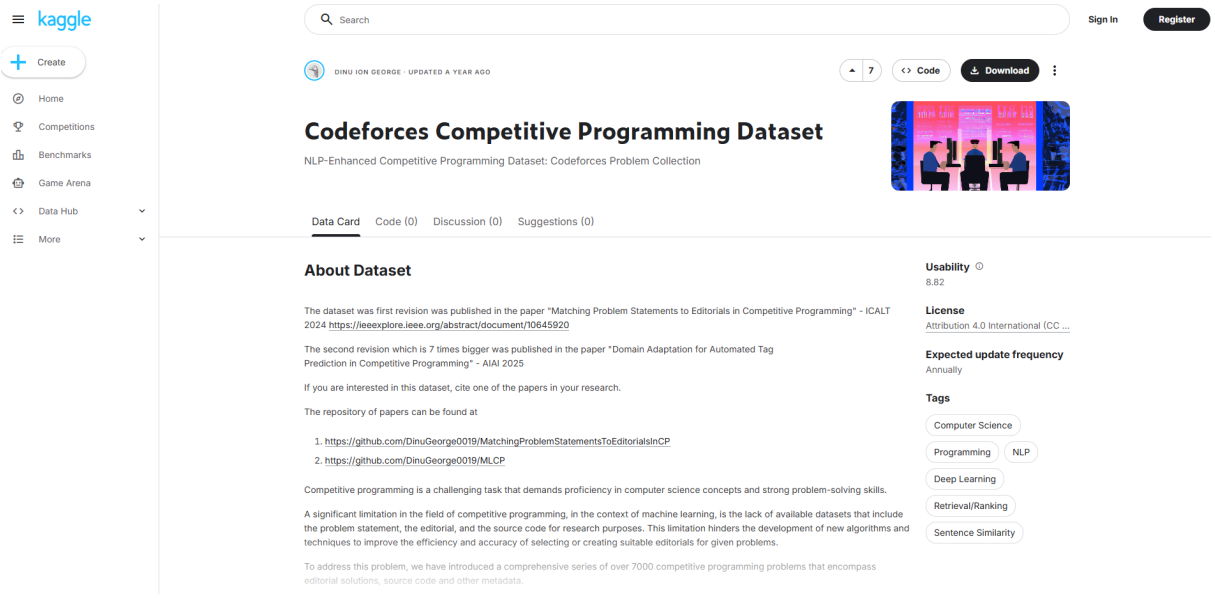

Figura 4 – Plataforma Kaggle - Fonte: (Kaggle, 2026)

2.2 Teorias Educacionais

Teorias educacionais oferecem diferentes formas de interpretar a prática, o desenvolvimento e as evidências produzidas durante a interação de um estudante com um ambiente digital. Entre elas, destacam-se o Construcionismo, a Zona de Desenvolvimento Proximal e a Avaliação Silenciosa.

2.2.1 Construcionismo

O Construcionismo, desenvolvido por Seymour Papert, defende que a aprendizagem pode ocorrer de forma mais sólida quando o estudante participa ativamente da construção de um artefato que possa ser testado (PAPERT, 1980). No contexto dos OJs, esse artefato pode ser o código-fonte. O competidor escreve uma solução, envia o código, recebe o resultado da avaliação e pode modificá-lo para uma nova tentativa.

2.2.2 Zona de Desenvolvimento Proximal

A Zona de Desenvolvimento Proximal, proposta por Vygotsky, representa a distância entre aquilo que o estudante já consegue realizar sozinho e aquilo que consegue realizar com orientação ou apoio (VYGOTSKY, 1978). Essa teoria destaca a importância de atividades que não sejam apenas repetições do que já é dominado, mas que também não estejam muito distantes das capacidades atuais do estudante.

2.2.3 Avaliação Silenciosa

A Avaliação Silenciosa, ou *Stealth Assessment*, integra a coleta de evidências ao próprio ambiente de atividade, sem exigir que o usuário realize um teste separado (SHUTE, 2011). As interações registradas durante uma atividade podem funcionar como sinais para estimar características relacionadas ao desempenho. Essas estimativas dependem das evidências disponíveis e não representam, isoladamente, uma avaliação completa do conhecimento.

2.3 Sistemas de Recomendação

Sistemas de Recomendação são ferramentas que ajudam os usuários a encontrar itens em ambientes com muitas opções (LEE; LAO, 2019; TOLEDO; MOTA; MARTíNEZ, 2018). Eles fazem parte dos mecanismos de filtragem de informação e podem ser aplicados em comércio eletrônico, turismo, redes sociais e ambientes de aprendizagem (TOLEDO; MOTA; MARTíNEZ, 2018; JULIANTI et al., 2022).

Em ambientes educacionais, esses sistemas podem recomendar materiais, atividades ou cursos de acordo com informações relacionadas aos estudantes (JULIANTI et al., 2022; MUEPU; WATANOBE; AMIN, 2025). Em uma plataforma de programação, o item recomendado pode ser um problema que o usuário ainda não resolveu. A seleção pode considerar o histórico de submissões, os assuntos praticados, a dificuldade e as características do problema.

2.3.1 Filtragem Colaborativa e Baseada em Conteúdo

A Filtragem Colaborativa produz recomendações a partir das interações de um conjunto de usuários (MUEPU; WATANOBE; AMIN, 2025). Ela pode procurar usuários com comportamentos semelhantes ou itens que receberam padrões semelhantes de interação (LEE; LAO, 2019). Uma dificuldade dessa abordagem é a partida a frio, ou *cold start*, que ocorre quando ainda existem poucas interações sobre um usuário ou item (CHIA; NAJAFABADI, 2022).

A Filtragem Baseada em Conteúdo compara as características dos itens com informações que representam o perfil do usuário (JULIANTI et al., 2022). Em problemas de programação, essas características podem incluir título, *tags*, dificuldade e conceitos relacionados à solução.

Uma técnica tradicional para representar textos é a Frequência do Termo–Inverso da Frequência nos Documentos (TF-IDF). Ela atribui peso a um termo de acordo com sua frequência em um documento e sua raridade no conjunto analisado (CHIA; NAJAFABADI, 2022). Sua forma básica pode ser representada por:

$$\text{tfidf}(x, y) = tf_{x,y} \times idf_x. \tag{1}$$

Por exemplo, se um termo aparece três vezes em um documento, de modo que $tf_{x,y} = 3$, e seu inverso da frequência nos documentos vale $idf_x = 2$, então:

$$\text{tfidf}(x, y) = 3 \times 2 = 6.$$

Depois da representação dos textos, a similaridade de cosseno pode ser utilizada para comparar dois vetores:

$$\text{sim}(A, B) = \frac{A \cdot B}{\|A\|\|B\|}. \tag{2}$$

Por exemplo, considere os vetores $A = (1, 0)$ e $B = (1, 1)$. O produto interno é igual a 1, enquanto as normas são 1 e $\sqrt{2}$. Assim:

$$\text{sim}(A, B) = \frac{1}{1 \times \sqrt{2}} \approx 0{,}707.$$

Quanto mais próximo o resultado estiver de 1, maior é a similaridade entre os vetores. Tanto o TF-IDF quanto os *embeddings* produzem representações vetoriais, mas de maneiras diferentes. O TF-IDF se baseia na frequência dos termos, enquanto os *embeddings* são produzidos por modelos que procuram representar relações semânticas presentes nos textos.

2.3.2 Recuperação Estruturada

A recuperação estruturada seleciona itens por meio de atributos explícitos e condições verificáveis. Em vez de comparar o significado de textos, ela aplica regras sobre campos conhecidos, como assunto, nível de dificuldade e identificador do item. Essa forma de recuperação está relacionada às abordagens baseadas em conteúdo, pois utiliza características cadastradas dos próprios itens (RICCI; ROKACH; SHAPIRA, 2015).

2.3.3 *Embeddings*, Busca Vetorial e Recuperação Semântica

Embeddings são representações numéricas que transformam textos em vetores. Textos que apresentam relações semânticas tendem a produzir vetores próximos no espaço matemático criado pelo modelo (LEWIS et al., 2020; MUEPU; WATANOBE; AMIN, 2025).

Os vetores podem ser comparados pela similaridade ou pela distância de cosseno. A distância é calculada por:

$$\text{dist}(A, B) = 1 - \text{sim}(A, B). \tag{3}$$

Usando a similaridade do exemplo anterior, a distância entre os vetores é:

$$\text{dist}(A, B) = 1 - 0{,}707 = 0{,}293.$$

Quanto menor a distância, maior é a proximidade estimada entre os vetores. A expressão busca vetorial baseada em *embeddings* identifica tanto a forma de representação quanto o mecanismo de busca utilizado. Essa especificação evita confusão com outras representações vetoriais, como o TF-IDF.

A recuperação é chamada de semântica quando a busca procura itens segundo relações representadas em seus conteúdos, e não somente pela igualdade literal entre palavras. Os *embeddings* não compreendem os textos da mesma maneira que uma pessoa; eles produzem uma representação matemática baseada nos padrões aprendidos pelo modelo. Por isso, a proximidade vetorial deve ser entendida como uma estimativa de relação semântica.

2.3.4 Banco Vetorial, PGVector e Índice HNSW

Um banco vetorial armazena *embeddings* e permite procurar os vetores mais próximos de uma consulta. O documento e a consulta precisam ser processados por modelos compatíveis para que seus vetores possam ser comparados. Na arquitetura RAG de Lewis et al. (LEWIS et al., 2020), esse índice externo funciona como uma memória não paramétrica consultada durante a recuperação.

O PGVector é uma extensão de código aberto para o PostgreSQL que adiciona o armazenamento de vetores e operações de busca por similaridade (PGVECTOR, 2026). Dessa forma, dados estruturados e *embeddings* podem permanecer no mesmo banco de dados.

Em catálogos grandes, comparar a consulta com todos os vetores pode exigir muitas operações. O Mundo Pequeno Navegável Hierárquico (HNSW), organiza os vetores em um grafo com camadas e permite realizar uma busca aproximada pelos vizinhos mais próximos (MALKOV; YASHUNIN, 2020). O índice procura reduzir o número de comparações necessárias sem alterar o significado dos *embeddings*.

2.3.5 Recuperação Híbrida

Abordagens híbridas combinam técnicas para aproveitar características complementares (RICCI; ROKACH; SHAPIRA, 2015). O significado do termo depende dos componentes reunidos. Ele pode representar, por exemplo, a combinação entre filtragem colaborativa e filtragem baseada em conteúdo ou a união entre filtros estruturados e uma busca semântica.

2.3.6 Seleção Determinística

Uma seleção é chamada de determinística quando as mesmas entradas e os mesmos parâmetros produzem a mesma saída. Em um sistema de recomendação, isso pode ser obtido por regras fixas de ordenação, desempate e distribuição dos itens. Essa forma de seleção facilita a reprodução de um procedimento, pois não depende de uma nova geração de texto a cada execução.

2.4 Inteligência Artificial Generativa e RAG

Esta seção introduz os fundamentos da Inteligência Artificial Generativa e a arquitetura RAG, conceitos centrais para compreender a etapa de curadoria e refinamento das listas de recomendação propostas neste estudo.

2.4.1 Modelos de Linguagem

A Inteligência Artificial Generativa reúne modelos capazes de produzir conteúdos a partir de padrões encontrados nos dados usados em seu treinamento. Quando a saída é textual, podem ser utilizados pela LLM. Esses modelos recebem uma entrada textual e produzem uma sequência de saída de acordo com o contexto fornecido (LEWIS et al., 2020).

O texto de entrada que descreve a tarefa e fornece o contexto ao modelo é chamado de *prompt*. Em um sistema de recomendação, uma LLM pode receber uma lista de itens previamente recuperados e escolher ou organizar parte deles. Essa atividade pode ser tratada como uma seleção, pois o modelo atua sobre um conjunto que já foi selecionado por outra etapa.

A temperatura é um parâmetro usado durante a geração para controlar a distribuição de probabilidade das próximas unidades da resposta. Valores menores tendem a concentrar a escolha nas opções mais prováveis, produzindo respostas menos variadas; valores maiores tendem a aumentar a variação (HOLTZMAN et al., 2020).

2.4.2 *Retrieval-Augmented Generation*

O RAG, combina uma etapa de busca com uma etapa generativa. Primeiro, um mecanismo de recuperação procura informações em uma fonte externa. Depois, os resultados recuperados são incluídos no contexto enviado à LLM (LEWIS et al., 2020).

Na proposta de Lewis et al. (LEWIS et al., 2020), o RAG combina a memória paramétrica da LLM com uma memória não paramétrica formada por um índice vetorial externo. O mecanismo consulta esse índice, recupera os documentos relacionados à entrada e fornece esse conteúdo ao modelo durante a geração.

O fluxo pode ser entendido em três partes:

1. a **consulta** representa a necessidade apresentada ao sistema;
2. a **recuperação** seleciona informações relacionadas à consulta;
3. a **geração** utiliza as informações recuperadas como contexto para produzir a saída.

Recuperação semântica e RAG não são sinônimos. Um sistema pode realizar busca vetorial baseada em *embeddings* e devolver os resultados sem chamar uma LLM. Nesse caso, existe recuperação semântica, mas não ocorre a etapa generativa necessária para caracterizar o fluxo completo de RAG.

2.5 Avaliação de Sistemas de Recomendação

A avaliação permite verificar se um sistema de recomendação atende ao objetivo para o qual foi desenvolvido. Como uma única medida não representa todos os aspectos de uma recomendação, diferentes métricas podem ser usadas para observar correspondência, alcance por usuário, dificuldade e diversidade (RICCI; ROKACH; SHAPIRA, 2015; MUEPU; WATANOBE; AMIN, 2025).

2.5.1 Avaliação *Offline* e Divisão Cronológica

A avaliação *offline* utiliza interações já registradas para construir perfis e comparar as recomendações geradas pelo sistema com uma parcela reservada dos dados (RICCI; ROKACH; SHAPIRA, 2015). Nesse formato de avaliação, uma parte do histórico do usuário é processada pelo sistema, enquanto a outra é mantida oculta para fins de comparação.

Quando a ordem cronológica das interações é preservada, os registros anteriores ao ponto de divisão formam o que se denomina **passado observado** (ou histórico de construção), enquanto os registros posteriores compõem o **futuro observado** (ou conjunto de avaliação). O passado observado é utilizado exclusivamente para a construção do perfil do usuário e como base para a etapa de recuperação e filtragem. Por sua vez, o futuro observado permanece isolado durante todo o processo de recomendação e é empregado unicamente na fase de validação dos resultados.

É importante destacar que o conjunto de avaliação contém apenas os itens que o usuário efetivamente escolheu e resolveu durante aquele período registrado. Portanto, um problema recomendado pelo sistema que não aparece no futuro observado não é, automaticamente, um erro ou um exercício inadequado; ele pode ser uma opção perfeitamente compatível com a proficiência do usuário, mas que simplesmente não foi escolhida por ele naquele intervalo de tempo. Da mesma forma, uma coincidência exata entre a recomendação e os problemas presentes no futuro observado demonstra a eficácia do sistema em antecipar e corresponder a parte do comportamento real do competidor.

2.5.2 Precisão no *Top-K*

A Precisão no *Top-K*, ou Precisão@K, mede a proporção de itens relevantes entre os K itens apresentados pelo sistema (RICCI; ROKACH; SHAPIRA, 2015; MUEPU; WATANOBE; AMIN, 2025). Para um usuário u, considere R_u^K como o conjunto dos K itens recomendados e T_u como o conjunto de itens usados para comparação:

$$\text{Precisao@K}(u) = \frac{\left|R_u^K \cap T_u\right|}{K}. \tag{4}$$

Por exemplo, se uma lista contém dez recomendações e duas delas aparecem entre os itens considerados relevantes, então:

$$\text{Precisao@10} = \frac{2}{10} = 0{,}20 = 20\%.$$

Quanto maior o resultado, maior é a proporção de recomendações que também aparecem no conjunto usado para comparação. A métrica considera a quantidade de coincidências, mas não informa diretamente quantos usuários receberam ao menos uma.

2.5.3 Taxa de Acerto por Usuário

A Taxa de Acerto por Usuário, também chamada de *Hit Rate*, mede a proporção de usuários que receberam pelo menos um item relevante na lista *Top-K* (RICCI; ROKACH; SHAPIRA, 2015). Considerando U como o conjunto de usuários:

$$\text{HitRate@K} = \frac{\left|\left\{u \in U : \left|R_u^K \cap T_u\right| > 0\right\}\right|}{|U|}. \tag{5}$$

Por exemplo, considere cinco usuários avaliados. Se dois deles receberam ao menos uma recomendação relevante, o cálculo é:

$$\text{HitRate@K} = \frac{2}{5} = 0{,}40 = 40\%.$$

Várias coincidências para o mesmo usuário não aumentam sua contribuição nessa métrica. Cada usuário recebe valor um quando existe ao menos uma coincidência e zero quando não existe.

2.5.4 Erro Absoluto Médio

O Erro Absoluto Médio (MAE), mede a diferença absoluta média entre valores previstos e valores usados para comparação. O valor absoluto impede que diferenças positivas e negativas se anulem (RICCI; ROKACH; SHAPIRA, 2015).

Para N observações, com valor previsto $\widehat{y}_i$ e valor usado para comparação y_i, o MAE é definido por:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N} |\widehat{y}_i - y_i| \,. \tag{6}$$

Por exemplo, considere três valores previstos, 800, 900 e 1000, e três valores usados para comparação, 900, 900 e 1100. Os erros absolutos são 100, 0 e 100. Assim:

$$\text{MAE} = \frac{|800-900| + |900-900| + |1000-1100|}{3} = \frac{200}{3} \approx 66{,}67.$$

Quanto menor o resultado, menor é a diferença média entre os valores comparados. A forma de definir $\widehat{y}_i$ e y_i depende do objetivo da avaliação.

2.5.5 Diversidade das Recomendações

A diversidade observa o quanto os itens recomendados diferem entre si. Ela complementa as métricas de coincidência, pois uma lista pode conter itens relacionados ao usuário, mas muito parecidos entre si (MUEPU; WATANOBE; AMIN, 2025). Precisão e diversidade representam aspectos diferentes: o aumento de uma não significa automaticamente o aumento da outra.

2.5.5.1 Diversidade Semântica Intra-Lista

A Diversidade Intra-Lista (ILD), mede a distância média entre todos os pares de itens de uma lista (SMYTH; MCCLAVE, 2001). Considere $R = \{r_1, r_2, \ldots, r_K\}$ como uma lista de K itens e $d(r_i, r_j)$ como a distância entre dois itens:

$$\text{ILD}(R) = \frac{2}{K(K-1)} \sum_{i=1}^{K-1} \sum_{j=i+1}^{K} d(r_i, r_j). \tag{7}$$

Por exemplo, uma lista com três itens possui três pares distintos. Se as distâncias desses pares forem 0,2, 0,4 e 0,6, então:

$$\text{ILD}(R) = \frac{2}{3(3-1)} (0{,}2 + 0{,}4 + 0{,}6) = 0{,}4.$$

Quando os itens são representados por *embeddings*, d pode ser a distância de cosseno definida na Equação 3. Quanto maior a ILD, maior é a diferença semântica média entre os itens da lista.

O número de pares depende do tamanho da lista. Cada item precisa ser comparado com os demais, mas cada par é contado apenas uma vez. Em uma lista com dez itens, a quantidade de pares diferentes é:

$$\binom{10}{2} = \frac{10 \times 9}{2} = 45.$$

O valor de 45 não significa que existam 45 problemas na lista. Ele representa as 45 combinações de dois problemas formadas pelos dez itens. A ILD calcula a distância de cada combinação e apresenta a média.

2.5.5.2 Variedade Global das Recomendações

Além da diversidade dentro de cada lista, pode-se contar quantos itens diferentes uma abordagem utiliza no conjunto completo de recomendações. Considerando a lista R_u^K de cada usuário u, essa contagem pode ser representada por:

$$\text{VariedadeGlobal} = \left| \bigcup_{u \in U} R_u^K \right|. \tag{8}$$

Por exemplo, considere três listas: $\{A, B, C\}$, $\{B, C, D\}$ e $\{A, D, E\}$. A união contém os itens $\{A, B, C, D, E\}$. Assim:

$$\text{VariedadeGlobal} = |\{A, B, C, D, E\}| = 5.$$

Essa contagem não corresponde à cobertura do catálogo, pois não é dividida pela quantidade total de itens disponíveis. Um valor maior indica somente que uma quantidade maior de identificadores distintos foi utilizada nas listas. Isoladamente, isso não demonstra maior precisão ou adequação.

2.6 Trabalhos Relacionados

Esta seção apresenta pesquisas sobre recomendação de problemas de programação em OJs e ambientes de aprendizagem. Primeiro, os estudos são organizados conforme seu foco principal. Depois, a relação entre eles e o sistema desenvolvido é discutida em uma única subseção comparativa.

2.6.1 Recomendação Baseada no Histórico dos Usuários

Toledo, Mota e Martínez (2018) propuseram um sistema de recomendação para OJs baseado em lógica *fuzzy*. A proposta utiliza o histórico de desempenho dos usuários e trata a incerteza presente nessas informações. O sistema foi avaliado com dados reais de um Juiz Online e apresentou seus melhores resultados para listas curtas.

Lee e Lao (2019) combinaram filtragem colaborativa, análise sequencial e um modelo em grafo para recomendar problemas. O algoritmo de Smith–Waterman foi utilizado para comparar sequências de atividades, enquanto o grafo representou relações entre os problemas. A avaliação foi realizada com dados do NTHU Online Judge.

2.6.2 Modelagem das Habilidades e da Progressão do Estudante

Zaffalon et al. (2022) apresentaram um sistema baseado no Modelo de Múltiplas Habilidades e Capacidades dos Estudantes (SMAS). O modelo combina a Teoria de Resposta ao Item com o sistema de classificação Elo que é um método matemático originalmente criado para o xadrez que calcula a proficiência e a força relativa dos competidores com base no resultado histórico de suas interações, para estimar diferentes habilidades do estudante. A proposta considera que um problema pode admitir caminhos de solução que exigem conjuntos próprios de conhecimentos. O sistema foi integrado a um Juiz Online e avaliado em uma turma de Algoritmos e Estruturas de Dados.

Ramesh e Kannimuthu (2023) propuseram um sistema sensível ao contexto que considera o nível atual e os objetivos de aprendizagem. O método identifica padrões de navegação entre níveis e combina filtragem colaborativa com filtragem baseada em conteúdo. A partir desses padrões, recomenda problemas do nível atual e de níveis seguintes.

2.6.3 Recomendação Baseada em Conteúdo e Abordagens Híbridas

Yoshimura et al. (2022) desenvolveram o WOJR, um sistema que recomenda problemas semelhantes a uma atividade considerada difícil. A comparação utiliza o conteúdo dos problemas e características do código-fonte de uma solução de referência. Em vez de fornecer a resposta da atividade, o sistema apresenta outros exercícios relacionados. A proposta foi avaliada por meio de uma intervenção em uma disciplina universitária.

Muepu, Watanobe e Amin (2025) desenvolveram um sistema baseado em conteúdo que representa problemas por características extraídas dos códigos-fonte. A análise inclui aspectos sintáticos, estruturais, estatísticos, semânticos e de complexidade. Os autores avaliaram combinações dessas características por métricas de relevância e diversidade.

Ansari et al. (2016) apresentaram o CodERS, um sistema híbrido desenvolvido para a plataforma CodeLearnr. A proposta combina diferentes fontes de informação sobre as atividades e o comportamento dos usuários para selecionar materiais de aprendizagem.

2.6.4 Comparação com os Trabalhos Relacionados

Os trabalhos analisados apresentam diferentes formas de lidar com a escolha de problemas em OJs. Toledo, Mota e Martínez (2018) tratam a incerteza presente no histórico, enquanto Lee e Lao (2019) exploram a sequência das atividades e as relações entre problemas. O sistema desenvolvido também utiliza o histórico dos usuários, mas preserva a ordem cronológica para construir um passado observado (histórico de construção) e um futuro observado (conjunto de avaliação).

A representação de diferentes habilidades se aproxima da proposta de Zaffalon et al. (2022), pois o perfil não é tratado como uma única capacidade geral. A diferença está na forma de representação: Zaffalon et al. (2022) utilizam habilidades associadas aos caminhos de solução, enquanto o sistema desenvolvido calcula um nível operacional separado para cada *tag* a partir dos problemas resolvidos no passado observado do usuário.

A consideração do nível atual e da progressão possui relação com o trabalho de Ramesh e Kannimuthu (2023). Entretanto, o estudo citado utiliza padrões de navegação encontrados entre estudantes, enquanto o sistema desenvolvido conta os problemas resolvidos em cada faixa de *rating* e ordena as *tags* da menor para a maior proficiência estimada. A primeira *tag* dessa ordem recebe uma posição adicional na lista final.

A utilização do conteúdo dos problemas se relaciona às propostas de Yoshimura et al. (2022) e Muepu, Watanobe e Amin (2025). Yoshimura et al. (2022) partem de uma atividade específica e procuram exercícios semelhantes, enquanto Muepu, Watanobe e Amin (2025) extraem características dos códigos-fonte. No sistema desenvolvido, os *embeddings* são produzidos a partir de uma descrição compacta dos problemas e usados somente depois da aplicação de filtros estruturados.

O termo híbrido também aparece no CodERS (ANSARI et al., 2016), mas representa outra combinação. No CodERS, diferentes fontes de informação sobre atividades e comportamento são integradas. No sistema desenvolvido, a recuperação híbrida combina filtros estruturados por *tag*, *rating* e histórico com busca vetorial baseada em *embeddings*.

A recuperação vetorial não forma sozinha o RAG. Na abordagem híbrida sem LLM, existe recuperação semântica, mas não existe geração. O fluxo completo de RAG ocorre na abordagem híbrida com LLM: o perfil produz uma consulta, o PGVector recupera os problemas por proximidade semântica entre os problemas elegíveis e parte desses resultados é incluída no *prompt*. A LLM realiza a etapa generativa ao devolver os identificadores escolhidos somente entre os problemas recuperados, seguindo a separação entre recuperação externa e geração apresentada por Lewis et al. (2020).

Por fim, a comparação experimental é realizada entre quatro abordagens aplicadas aos mesmos usuários: estruturada sem LLM, estruturada com LLM, híbrida sem LLM e híbrida com LLM. Essa organização permite observar separadamente as mudanças associadas à busca vetorial baseada em *embeddings* e à seleção da LLM.

A Tabela 1 sintetiza as principais características dos trabalhos relacionados e sua relação com esta pesquisa.

Tabela 1 – Comparação com os trabalhos relacionados

Trabalho	**Informações utilizadas**	**Método principal**	**Relação com esta pesquisa**
Toledo, Mota e Martínez (2018)	Histórico de desempenho dos usuários	Lógica *fuzzy*	Ambos usam o histórico. Aqui, preserva-se a ordem cronológica e o perfil é separado por *tag*.
Lee e Lao (2019)	Sequência de atividades e relação de problemas	Filtro colaborativo, sequência e modelo em grafo	Ambos usam histórico. Aqui, a recomendação foca em *tags*, *rating* e busca semântica.
Zaffalon et al. (2022)	Habilidades e caminhos de solução	Modelo SMAS, Teoria de Resposta ao Item e Elo	Ambos modelam múltiplas habilidades. Aqui, elas são representadas pelas *tags* da plataforma.
Ramesh e Kannimuthu (2023)	Nível atual, objetivos e padrões de navegação	Filtragem colaborativa e baseada em conteúdo	Ambos usam níveis por assunto. Aqui, o nível depende da quantidade resolvida por *rating*.
Yoshimura et al. (2022)	Conteúdo dos problemas e código de referência	Recomendação de exercícios semelhantes	Ambos relacionam problemas. Aqui, a comparação usa *embeddings* após filtros estruturados.
Muepu, Watanobe e Amin (2025)	Características dos códigos-fonte	Recomendação em conteúdo com análise multifacetada	Ambos avaliam diversidade. Aqui, os vetores representam metadados textuais compactos.
Ansari et al. (2016)	Atividades e comportamento dos usuários	Sistema híbrido CodERS	Ambos são híbridos. Aqui, isso significa juntar filtros estruturados e busca semântica.
Esta pesquisa	Histórico cronológico, *tags* e *rating*	Perfil multitag, filtros, busca vetorial e seleção via LLM	Gera listas de dez problemas; avalia quatro abordagens de forma *offline* e temporal.

Fonte: Autor

3 Metodologia

Este capítulo apresenta o método utilizado para construir e avaliar o sistema de recomendação. Primeiro, são descritos a caracterização da pesquisa, a preparação da base de dados e a arquitetura do sistema proposto. Em seguida, detalha-se o desenho experimental e as métricas de avaliação escolhidas para validar as hipóteses do trabalho.

3.1 Arquitetura e Preparação dos Dados

Esta seção descreve a etapa inicial do método e engloba a caracterização da pesquisa e os procedimentos de coleta e tratamento das informações. O texto detalha os passos para a limpeza da base, a separação cronológica do histórico dos usuários e a construção do perfil de proficiência. Essas etapas estabelecem a base de dados que alimentará os modelos de recomendação.

3.1.1 Caracterização da Pesquisa

Esta pesquisa possui caráter aplicado, pois desenvolve e avalia um sistema de recomendação de problemas de programação. A avaliação é quantitativa e *offline*, baseada em registros públicos da plataforma Codeforces. As recomendações geradas pelo sistema são comparadas com os problemas efetivamente resolvidos pelos competidores no futuro observado.

Os mesmos usuários foram processados pelas quatro abordagens do sistema. Esse procedimento pareado permite comparar rigorosamente a recuperação estruturada e a recuperação híbrida, com e sem a participação da LLM, mantendo a mesma amostra, o mesmo histórico de construção do perfil (passado observado), o mesmo conjunto de avaliação (futuro observado) e o mesmo tamanho de lista de recomendações.

3.1.2 Fonte e Preparação dos Dados

Os dados foram obtidos por meio da API pública do Codeforces (Codeforces, 2026). Foram coletados o identificador e o *rating* dos usuários, as submissões com veredito *Accepted* e os dados dos problemas, incluindo identificador, título, *rating* e *tags*. As submissões aceitas foram consideradas em ordem cronológica para representar o histórico de resolução de cada usuário.

O catálogo armazenado no PostgreSQL também foi enriquecido com um arquivo preparado a partir do *Codeforces Competitive Programming Dataset*, disponibilizado no Kaggle por Dinu (DINU; MIHAESCU; REBEDEA, 2024; DINU, 2025). A integração entre as

duas fontes foi realizada pelo identificador oficial do problema: o campo `short_id` do arquivo complementar foi associado ao identificador obtido pela API do Codeforces. Com essa operação, foram armazenados, quando disponíveis, o enunciado, a solução editorial e a dica técnica. Assim, os dados de usuários, submissões, *ratings* e *tags* vieram da API, enquanto os campos textuais complementares vieram do arquivo derivado da base do Kaggle.

A coleta inicial selecionou 100 usuários com *rating* entre 600 e 1400, com o objetivo de focar no público iniciante da plataforma (`newbie e pupil`). Além disso, exigiu-se que cada usuário possuísse, no mínimo, 100 submissões aceitas. Esse valor foi estabelecido por decisão de projeto para garantir um volume histórico suficiente: era necessário ter dados bastantes para construir um perfil sólido do competidor e, ao mesmo tempo, deixar uma margem de problemas futuros para avaliar as recomendações. Durante a etapa de geração de resultados, devido às restrições operacionais de limite de *tokens* e cotas de uso na *API* da LLM, o processamento em lote de todos os usuários atingiu seu limite. Por esse motivo, a amostra final foi ajustada de 100 para 85 usuários, garantindo assim que a execução do experimento ocorresse de forma completa e pareada para as quatro abordagens avaliadas.

Na amostra final, foram registrados 29.289 *Accepted*, com média de 344,58 por usuário, mínimo de 101 e máximo de 1.864. Esses valores representam registros de submissão, e não problemas necessariamente diferentes.

3.1.2.1 Submissões repetidas e problemas distintos

O Codeforces armazena cada envio de código como uma submissão independente. Assim, o mesmo usuário pode receber mais de um *Accepted* para o mesmo problema, por exemplo, ao enviar outra implementação, testar uma linguagem diferente ou apresentar uma nova solução depois de já ter resolvido o exercício. Os identificadores das submissões são diferentes, mas o identificador do problema permanece o mesmo.

Para esta pesquisa, o interesse está em saber se o usuário resolveu determinado problema. Por isso, vários *Accepted* do mesmo usuário para o mesmo problema não foram contados como vários exercícios resolvidos. Os registros foram agrupados pelo identificador oficial do problema, preservando a primeira ocorrência aceita no histórico do usuário.

A escolha metodológica pela *primeira* ocorrência aceita fundamenta-se no rastreamento da trajetória de aprendizagem do competidor. O primeiro *Accepted* marca o momento temporal exato em que o usuário comprovou o domínio sobre o conhecimento exigido por aquele desafio. Submissões posteriores do mesmo problema costumam representar otimizações de complexidade (tempo e memória), traduções para outras linguagens de programação ou simples revisões.

Ademais, essa decisão tem impacto direto na preservação da divisão cronológica do experimento. Caso a *última* ocorrência fosse utilizada, a linha do tempo do usuário

seria artificialmente alterada: um problema já dominado nas etapas iniciais (que deve pertencer ao histórico de construção do perfil) poderia ser deslocado para o final do histórico (conjunto de avaliação) apenas devido a uma revisão tardia. Isso distorceria o cálculo do nível de proficiência da *tag* no momento em que as recomendações são geradas.

Após essa operação, os 29.289 registros aceitos resultaram em 27.273 pares distintos entre usuário e problema. Foram removidos, portanto, 2.016 registros repetidos, equivalentes a 6,88% dos *Accepted*. Em 79 dos 85 usuários existia pelo menos uma repetição. Essa deduplicação e a manutenção da primeira ocorrência impedem que vários códigos aceitos para o mesmo exercício aumentem artificialmente a frequência de uma *tag*, a progressão de nível ou corrompam a linha do tempo da avaliação.

3.1.2.2 Divisão cronológica 30/70

Depois da limpeza dos dados duplicados, o histórico de problemas distintos de cada usuário foi separado cronologicamente em dois conjuntos:

- ❑ aproximadamente os 30% iniciais formaram o histórico de construção do perfil (passado observado), utilizado exclusivamente para calcular as proficiências e gerar as recomendações.
- ❑ aproximadamente os 70% finais formaram o conjunto de avaliação (futuro observado), que permaneceu isolado e foi utilizado para verificar a correspondência (acertos) das recomendações.

Depois da remoção das repetições e sobreposições, o passado observado reuniu 8.218 problemas, enquanto o futuro observado reuniu 19.055. Assim, o valor absoluto correspondente aos 30,13% é 8.218 de um total de 27.273 pares distintos entre usuário e problema:

$$\frac{8\,218}{27\,273} \times 100 = 30,13\%. \tag{9}$$

O número de problemas distintos por usuário variou de 100 a 1.713, com média de 320,86. No passado observado, a média foi de 96,68 problemas por usuário, com mínimo de 30 e máximo de 527. No futuro observado, a média foi de 224,18 problemas, com mínimo de 68 e máximo de 1.186.

Foram retirados do passado observado os problemas que também apareciam no futuro observado, preservando no passado observado a ocorrência anterior. Essa correção evita considerar como acerto uma recomendação de um problema que o sistema já sabia ter sido resolvido. O futuro observado não participa da seleção das *tags*, do cálculo do nível nem da recuperação dos problemas.

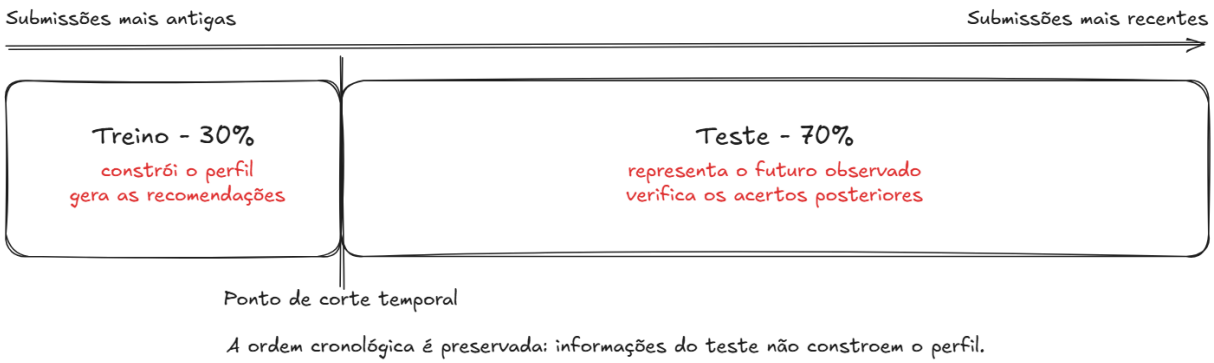

Figura 5 – Divisão cronológica do histórico em passado e futuro - Fonte: Autor

3.1.3 Construção do Perfil por *Tags*

O perfil do usuário é construído somente a partir do passado observado. Cada problema pode possuir várias *tags*; por isso, um mesmo problema pode contribuir para mais de um assunto na contagem do perfil. Dos 8.218 problemas do passado, 8.042 possuíam metadados no catálogo local e 176 não puderam participar da contagem das *tags*. Considerando os problemas com metadados, os usuários apresentaram em média 17,32 *tags* distintas. O mínimo foi 9, a mediana foi 16 e o máximo foi 32.

Embora exista uma quantidade ampla de assuntos, as três *tags* mais frequentes cobriram, em média, 87,33% dos problemas do passado mapeados. Para esse cálculo, um problema é considerado coberto quando possui pelo menos uma das três *tags* selecionadas. Por esse motivo, neste trabalho, foram consideradas três *tags* como um ponto de equilíbrio entre representar os assuntos principais do usuário e manter uma lista final curta. A utilização de três assuntos também permite distribuir as dez recomendações em cotas de quatro, três e três problemas.

A estimativa de nível é calculada separadamente para cada *tag*. O nível inicial foi definido como 800, correspondente ao menor *rating* normalmente utilizado no catálogo do Codeforces. Neste trabalho, a autora definiu seis problemas distintos como o limiar operacional para avançar 100 pontos em uma *tag*. Essa quantidade é um parâmetro operacional da proposta, utilizado para calcular a progressão do nível por *tag*.

O sistema conta os problemas resolvidos no passado em cada combinação de *tag* e *rating*. Quando existem pelo menos seis problemas no nível atual, o nível aumenta em 100 pontos e a verificação é repetida. Por exemplo, seis problemas de *math* com *rating* 800 fazem o indicador dessa *tag* avançar para 900. Por representar uma regra computacional criada para esta pesquisa, o resultado é chamado de nível operacional observado por *tag*.

Depois do cálculo, as três *tags* são ordenadas da menor para a maior proficiência. Primeiro é considerado o nível atual e, em caso de empate, é usada a quantidade resolvida

naquele nível. Essa ordem orienta a distribuição das recomendações e oferece uma vaga adicional para a primeira *tag*.

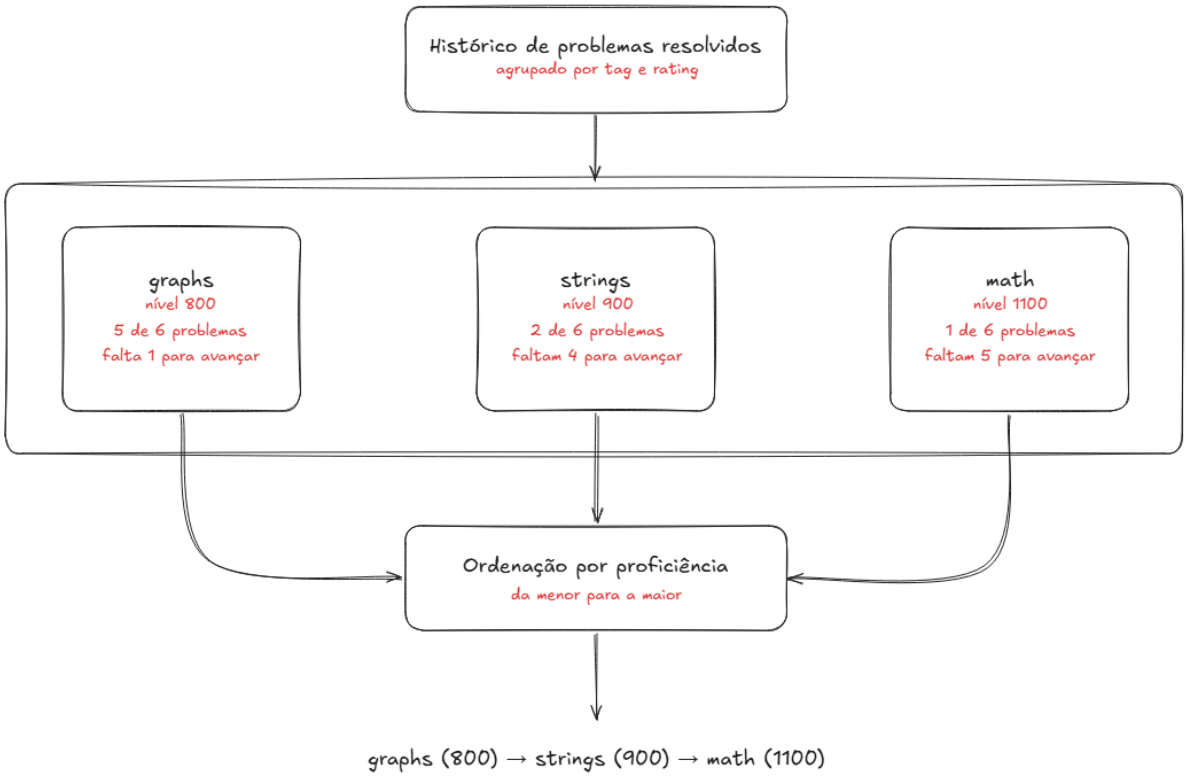

Figura 6 – Exemplo da construção e da ordenação do perfil multitag - Fonte: Autor

3.1.4 Representação dos Problemas por *Embeddings*

No sistema implementado, a recuperação semântica é realizada por uma busca vetorial baseada em *embeddings*. O *embedding* é a representação numérica produzida para cada texto, enquanto a busca vetorial é o mecanismo que compara essas representações e localiza os vetores mais próximos.

Cada problema foi representado por um documento textual compacto formado por título, *tag* usada na recuperação, conjunto de *tags*, *rating* e dica técnica, quando disponível. O enunciado não participa da representação final.

O modelo `nomic-embed-text` transforma cada documento em um *embedding* de 768 dimensões. Um *embedding* é uma lista de números que representa o texto em um espaço matemático. O mesmo modelo também transforma a consulta produzida para o usuário em um vetor com a mesma dimensão.

Os vetores foram armazenados no PostgreSQL por meio da extensão PGVector. Ao final da rotina, o campo `indexedDocuments` do relatório de indexação registrou 20.634 documentos vetoriais. Esse número não representa 20.634 problemas diferentes: foi criado um documento para cada combinação entre problema e *tag*. Assim, um problema com

três *tags* gera três documentos relacionados ao mesmo identificador. Essa organização permite aplicar um filtro exato de *tag* durante a busca vetorial.

O índice HNSW foi usado para acelerar a localização dos vetores mais próximos. De forma simplificada, ele organiza os vetores em uma estrutura de conexões e evita comparar a consulta com todos os documentos um a um. A distância de cosseno foi utilizada para medir a diferença de direção entre o vetor da consulta e o vetor de cada documento; quanto menor a distância, maior a proximidade semântica estimada. O valor de 20.634 pode ser conferido pela quantidade de registros da tabela vetorial e corresponde à soma das combinações problema–*tag* efetivamente indexadas.

3.1.5 Recuperação dos Problemas

3.1.5.1 Recuperação estruturada

A recuperação estruturada utiliza regras explícitas. Para cada *tag*, o sistema procura problemas que:

- ❏ possuam a *tag* analisada;
- ❏ não tenham sido resolvidos no passado observado;
- ❏ estejam entre o nível atual e o nível atual acrescido de 100 pontos.

Nessa abordagem, são recuperados diretamente até 15 problemas por *tag*, ordenados pelo menor *rating* e, em seguida, pelo identificador oficial. Como o mesmo problema pode possuir mais de uma das três *tags*, identificadores repetidos são removidos durante a combinação das listas.

3.1.5.2 Recuperação híbrida e semântica

A recuperação híbrida começa com os mesmos filtros. Para cada *tag*, são obtidos até 50 problemas estruturalmente compatíveis. Esse conjunto forma uma lista de elegibilidade: a busca vetorial não pode introduzir um problema que esteja fora das regras de assunto, dificuldade ou do passado observado.

Em seguida, é criada uma consulta textual com a *tag*, o nível atual, a quantidade resolvida nesse nível, a quantidade que falta para avançar e os títulos de até cinco problemas recentes daquele assunto no passado observado. A consulta é transformada em *embedding* e comparada com os vetores armazenados no PGVector.

Os problemas elegíveis são reorganizados pela proximidade vetorial, e até 15 problemas por *tag* permanecem na lista recuperada. Quando a busca vetorial não fornece a quantidade necessária, o sistema completa a lista com problemas da ordenação estruturada.

3.1.5.3 Combinação entre as *tags*

As listas das três *tags* são combinadas de forma alternada. O sistema adiciona o primeiro problema da primeira *tag*, depois o primeiro da segunda, o primeiro da terceira e repete o processo com as próximas posições. Esse procedimento evita que apenas um assunto ocupe o início da lista.

3.1.6 Seleção com a LLM

Nas abordagens com LLM, o modelo `gemma4:31b-cloud` não recebe os 50 problemas iniciais de cada *tag*. Antes da chamada, o sistema separa no máximo 15 problemas recuperados no total, distribuídos entre as três *tags*. Para cada problema são enviados identificador, título, *tag* de destino, *rating*, valor de similaridade e conjunto de *tags*.

O *prompt* de sistema foi construído para limitar a função da LLM à escolha dos identificadores dos problemas já recuperados na etapa anterior. Para garantir a transparência da etapa generativa, o conteúdo exato instruído ao modelo foi: "Você é o curador final de um sistema RAG educacional. Selecione exatamente 10 problemas somente dentre os problemas fornecidos. Priorize as tags na ordem de menor para maior proficiência e respeite a dificuldade alvo. Distribua as recomendações entre todas as tags. Responda apenas com IDs oficiais separados por espaço, sem explicações e sem inventar IDs. Perfil: [perfil calculado do usuário]". Detalhes adicionais sobre a injeção das variáveis e o *prompt* do usuário encontram-se no Apêndice A.5.

A temperatura foi configurada como zero. A temperatura controla a aleatoriedade usada na escolha dos próximos elementos da resposta: valores maiores permitem saídas mais variadas, enquanto valores baixos favorecem respostas mais estáveis. O valor zero foi usado para aumentar a reprodutibilidade. Ele não elimina sozinho a possibilidade de IDs incorretos. A proteção efetiva ocorre porque o sistema compara a resposta com a lista permitida, descarta qualquer identificador externo e completa eventuais faltas de forma determinística.

A lista final sempre possui dez problemas. As cotas esperadas são quatro problemas para a primeira *tag* e três para cada uma das outras duas. Portanto, os números possuem funções diferentes:

- 50: limite inicial de problemas estruturados por *tag* na recuperação híbrida;
- 15 por *tag*: limite depois da ordenação semântica;
- 15 no total: limite enviado à LLM;
- 10: quantidade final apresentada como recomendação.

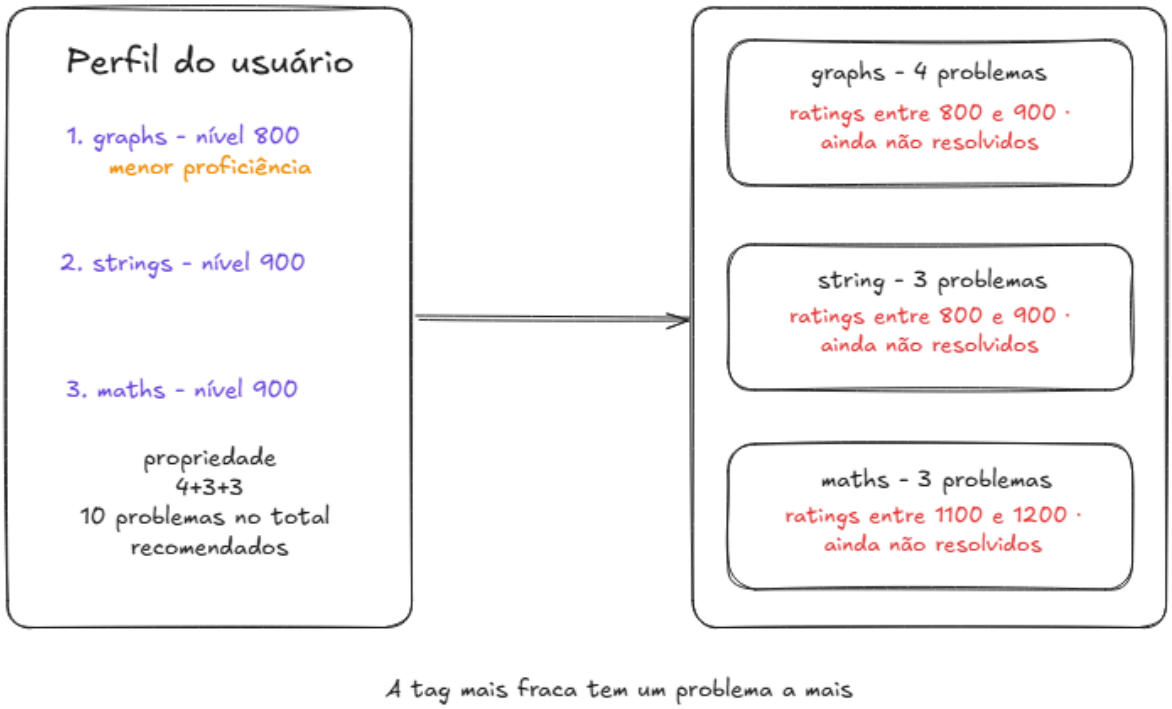

Figura 7 – Exemplo de distribuição dos dez problemas entre as três *tags* do usuário.

A escolha da LLM neste caso, o gemma4:31b-cloud acessado via Ollama) foi baseada em critérios técnicos voltados para a arquitetura RAG. O modelo demonstrou alta capacidade de seguir instruções estritas de formatação, requisito fundamental para que a saída gerasse estritamente os identificadores (IDs) solicitados, sem incluir textos adicionais ou alucinações que quebrassem o fluxo determinístico da aplicação. Além disso, o modelo apresenta uma janela de contexto e capacidade de raciocínio adequadas para processar os metadados dos 15 problemas fornecidos no prompt e realizar a curadoria de forma eficaz.

3.1.7 Abordagens Comparadas

1. **Estruturada sem LLM**: filtros por *tag* e *rating*, seguidos de seleção determinística;
2. **Estruturada com LLM**: filtros estruturados e seleção final pela LLM;
3. **Híbrida sem LLM**: filtros estruturados, busca vetorial baseada em *embeddings* e seleção determinística;
4. **Híbrida com LLM**: filtros, busca vetorial baseada em *embeddings* e seleção da LLM, formando o fluxo de RAG avaliado.

Na abordagem híbrida com LLM, a recuperação corresponde à busca dos problemas no PGVector; o aumento de contexto ocorre quando os problemas recuperados são incluídos no *prompt*; e a geração corresponde à lista de IDs devolvida pela LLM. A abordagem

híbrida sem LLM utiliza recuperação semântica, mas não executa a etapa generativa do RAG.

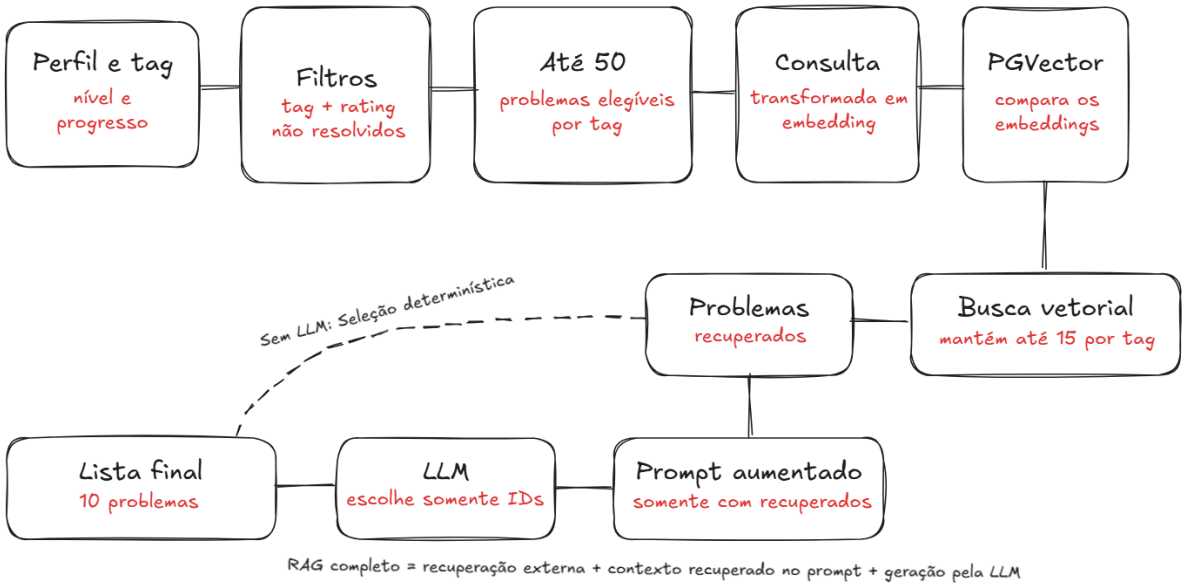

Figura 8 – Fluxo geral de construção, recomendação e avaliação do sistema - Fonte: Autor

3.2 Desenho Experimental e Métricas

3.2.1 Parâmetros do Experimento

Para garantir a transparência e a reprodutibilidade da pesquisa, os parâmetros adotados para a coleta de dados, cálculo de níveis de proficiência, filtragem e limites de recuperação vetorial e geração foram consolidados. A Tabela 2 sintetiza as principais estatísticas da amostra e as constantes operacionais aplicadas durante os experimentos em todas as abordagens avaliadas.

Tabela 2 – Dados e parâmetros utilizados no experimento

Informação	**Valor**
Usuários na amostra final	85
Registros de submissões *Accepted*	29.289
Média de *Accepted* por usuário	344,58
Mínimo e máximo de *Accepted* por usuário	101 e 1.864
Problemas distintos por usuário, em média	320,86
Mínimo e máximo de problemas distintos	100 e 1.713
Problemas distintos no passado observado	8.218
Problemas do passado observado com metadados de *tags*	8.042
Problemas do passado observado sem metadados de *tags*	176
Média de problemas do passado observado por usuário	96,68
Mínimo e máximo de problemas do passado observado	30 e 527
Problemas distintos no futuro observado	19.055
Média de problemas do futuro observado por usuário	224,18
Mínimo e máximo de problemas do futuro observado	68 e 1.186
Média de *tags* distintas por usuário	17,32
Mínimo e máximo de *tags* distintas	9 e 32
Tags utilizadas no perfil	3
Cobertura média das três *tags*	87,33%
Nível inicial e passo entre níveis	800 e 100
Problemas necessários para avançar de nível	6
Documentos armazenados no PGVector	20.634
Dimensão dos *embeddings*	768
Problemas estruturados por *tag* na recuperação híbrida	50
Problemas mantidos por *tag* após a busca vetorial	15
Problemas enviados à LLM	15
Problemas na lista final	10
Temperatura da LLM	0

3.2.2 Implementação

O sistema foi desenvolvido em Java com Spring Boot e Spring AI. O catálogo de problemas foi armazenado no PostgreSQL, e o PGVector foi utilizado para a busca vetorial baseada em *embeddings*. O Ollama forneceu acesso ao modelo de *embedding* e à LLM.

Os resultados foram armazenados em arquivos Valores Separados por Vírgulas (CSV). Cada linha contém a abordagem utilizada, as *tags* ordenadas, o perfil calculado, a quantidade de problemas recuperados, os dez identificadores recomendados, os acertos e a indicação de uso da LLM. Os trechos de código responsáveis pelas estatísticas da amostra, pelo cálculo do nível, pela recuperação semântica, pelo cálculo do MAE e pela construção dos *prompts* são apresentados no Apêndice A.

3.2.3 Desenho Experimental

Foi realizada uma avaliação *offline* e pareada. Os mesmos 85 usuários foram processados pelas quatro abordagens, e cada abordagem produziu uma lista de dez problemas por usuário. O número total de listas foi calculado por:

$$85 \text{ usuários} \times 4 \text{ abordagens} = 340 \text{ listas}. \tag{10}$$

Como cada lista contém dez problemas, foram produzidas 3.400 recomendações. Para cada abordagem foram geradas 85 listas e 850 recomendações.

Uma recomendação foi considerada um acerto exato quando o identificador oficial do problema também apareceu no conjunto de avaliação do mesmo usuário, isto é, em seu futuro observado. A comparação quantitativa principal ocorre entre as quatro abordagens do próprio sistema, sendo a recuperação estruturada a referência interna para avaliar o procedimento híbrido. Os trabalhos discutidos no capítulo de fundamentação e os trabalhos relacionados são comparados de forma conceitual, pois utilizam bases de dados, públicos e protocolos de avaliação diferentes. Por esse motivo, seus valores numéricos não são tratados como uma linha de base (*baseline*) diretamente equivalente.

3.2.4 Precisão@10 e *Hit Rate@10*

A Precisão@10 mede a proporção de acertos entre os dez problemas de cada lista. No resultado agregado, o total de acertos é dividido por 850, pois cada abordagem contém 85 usuários e dez recomendações:

$$\text{Precisao@10} = \frac{\text{total de acertos da abordagem}}{85 \times 10}. \tag{11}$$

O *Hit Rate@10* mede quantos usuários receberam pelo menos um problema presente no futuro observado:

$$\text{HitRate@10} = \frac{\text{usuários com pelo menos um acerto}}{85}. \tag{12}$$

Assim, o número absoluto 38 significa que 38 dos 85 usuários receberam pelo menos um problema que também apareceu em seu futuro observado. Esse total corresponde a:

$$\frac{38}{85} \times 100 = 44,71\%. \tag{13}$$

3.2.5 Erro Absoluto Médio

O MAE representa o erro absoluto médio. Como o sistema produz uma lista de problemas, e não uma única previsão numérica, a métrica foi adaptada para comparar duas médias de dificuldade. A média de *rating* da lista recomendada funciona como o

valor produzido pelo sistema, enquanto a média de *rating* do futuro observado funciona como valor de comparação.

Para cada usuário u, calcula-se primeiro a média de *rating* dos dez problemas recomendados, denotada por $\overline{r}_u^{rec}$, e a média de *rating* de todos os problemas com dificuldade conhecida no futuro observado, independentemente da *tag*, denotada por $\overline{r}_u^{futuro}$. O erro do usuário é:

$$e_u = \left| \overline{r}_u^{rec} - \overline{r}_u^{futuro} \right|. \tag{14}$$

O resultado final é a média dos erros dos 85 usuários:

$$\mathrm{MAE}_{rating} = \frac{1}{85} \sum_{u=1}^{85} e_u. \tag{15}$$

Todos os 85 usuários possuíam valores de *rating* suficientes para calcular as duas médias. Portanto, nenhuma observação foi retirada do cálculo final do MAE.

Por exemplo, se a média dos dez problemas recomendados para um usuário for 950 e a média dos problemas de seu futuro observado for 1100, o erro desse usuário será:

$$e_u = |950 - 1100| = 150 \text{ pontos de } \textit{rating}. \tag{16}$$

Um MAE menor indica que a dificuldade média recomendada ficou mais próxima da dificuldade média encontrada no futuro observado. Essa adaptação mede a proximidade entre médias de dificuldade. Ela complementa a Precisão@10 e o *Hit Rate@10* ao observar apenas a diferença de dificuldade. O código utilizado nesse cálculo é apresentado no Apêndice A.

3.2.6 Diversidade Semântica Intra-Lista

A ILD calcula a distância média de cosseno entre os pares de problemas de uma mesma lista. Como cada lista possui dez problemas, são comparados:

$$\binom{10}{2} = \frac{10 \times 9}{2} = 45 \text{ pares}. \tag{17}$$

Os 45 pares representam todas as combinações de dois problemas formadas pelos dez itens da lista, sem repetir a ordem. Para cada par, foi calculada a distância de cosseno entre os *embeddings* armazenados no PGVector para a combinação entre problema e *tag* de destino. A diversidade da lista corresponde à média dessas 45 distâncias, e a diversidade de cada abordagem corresponde à média das 85 listas produzidas. Na execução, os *embeddings* foram encontrados para os dez problemas de todas as 340 listas; assim, cada lista contribuiu com exatamente 45 distâncias.

Por exemplo, em uma lista reduzida de três problemas, existem três pares. Se suas distâncias forem 0,2, 0,4 e 0,6, a diversidade da lista será:

$$\text{ILD} = \frac{0,2 + 0,4 + 0,6}{3} = 0,4. \tag{18}$$

Valores maiores indicam que os vetores dos problemas de uma lista são, em média, mais diferentes entre si. Essa medida é complementar às métricas de acerto. Além da ILD, foi calculada a variedade global de cada abordagem. Essa medida corresponde à quantidade de identificadores distintos encontrados na união das 85 listas, ou seja, entre as 850 recomendações produzidas por uma abordagem. Ela não representa cobertura do catálogo, pois não divide essa quantidade pelo total de problemas disponíveis.

4 Resultados e Discussão

Este capítulo apresenta e discute os resultados obtidos a partir da execução das quatro abordagens de recomendação. Inicialmente são reportados os dados gerais de acerto e de erro, bem como a adequação de dificuldade. Na sequência o texto analisa o impacto das abordagens na diversidade semântica das listas. Por fim os resultados são discutidos em resposta às hipóteses de pesquisa formuladas.

4.1 Execução e Resultados

De acordo com o método descrito na Seção 3.2, as quatro abordagens foram executadas sobre os mesmos usuários, considerando o mesmo passado e futuro observados. Esta seção apresenta primeiro os dados gerais da execução e, depois, os resultados obtidos em cada métrica.

4.1.1 Execução das Quatro Abordagens

As quatro abordagens produziram dez problemas distintos para cada usuário. Nas duas abordagens com LLM, o registro de execução confirmou o uso do modelo nos 85 casos.

Antes da seleção final, a recuperação estruturada produziu em média 38,59 problemas diferentes por usuário, com valores entre 31 e 45. A recuperação híbrida produziu em média 39,58, com valores entre 33 e 45. O máximo teórico é 45 porque são mantidos até 15 problemas para cada uma das três *tags*. O total pode ser menor quando o mesmo problema aparece em mais de uma *tag* e seu identificador é deduplicado.

Nas abordagens com LLM, somente 15 desses problemas recuperados foram incluídos no *prompt*. A resposta foi validada e reduzida à lista final de dez recomendações. Portanto, “15” representa o limite de entrada da LLM, enquanto “10” representa o tamanho da recomendação avaliada.

4.1.2 Resultados de Acerto

Os registros originais do Codeforces podem conter vários *Accepted* do mesmo usuário para o mesmo problema, pois cada novo código enviado gera uma submissão independente. Essas ocorrências não foram tratadas como exercícios diferentes: antes da divisão da base, elas foram agrupadas pelo identificador oficial do problema. Assim, um problema repetido no histórico pode representar códigos diferentes, mas contribui apenas uma vez para o passado observado, para a contagem das *tags* e para a avaliação.

Tabela 3 – Resultados das quatro abordagens

Abordagem	**Usuários com acerto**	***Hit Rate@10***	**Acertos**	**Precisão@10**	**MAE**
Estruturada sem LLM	8	9,41%	11	1,29%	156,99
Estruturada com LLM	9	10,59%	13	1,53%	156,99
Híbrida sem LLM	38	44,71%	73	8,59%	157,70
Híbrida com LLM	38	44,71%	68	8,00%	157,46

Na abordagem estruturada sem LLM, 8 dos 85 usuários receberam ao menos um acerto. Com a LLM, esse número passou para 9. Nas duas abordagens híbridas, 38 usuários receberam ao menos um problema que também estava em seu conjunto de avaliação (futuro observado).

A quantidade de acertos considera todas as coincidências das listas. A abordagem híbrida sem LLM produziu 73 acertos entre 850 recomendações, resultando em Precisão@10 de 8,59%. A híbrida com LLM produziu 68 acertos, correspondentes a 8,00%.

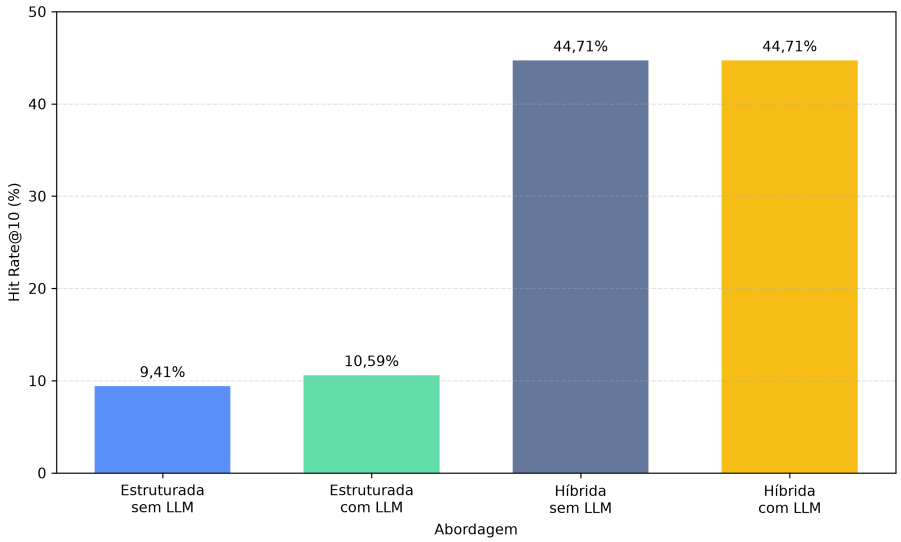

Figura 9 – Comparação do *Hit Rate@10* entre as quatro abordagens - Fonte: Autor

4.1.3 Resultados do MAE

Os valores do MAE ficaram entre 156,99 e 157,70 pontos. A diferença máxima entre as quatro abordagens foi inferior a um ponto. Portanto, essa medida não diferenciou de forma relevante as abordagens.

Esse comportamento é coerente com a implementação, pois todas as formas de recuperação utilizam filtros de dificuldade antes da seleção final. A busca vetorial e a LLM alteram quais problemas são escolhidos dentro do conjunto permitido, mas não removem os limites de *rating*.

4.1.4 Adequação da Dificuldade

Todas as 3.400 recomendações possuíam *rating* conhecido e permaneceram entre o nível atual calculado para a *tag* e esse nível acrescido de 100 pontos. Esse resultado mostra que a busca vetorial e a LLM não ultrapassaram as restrições aplicadas pela recuperação estruturada.

Na recuperação estruturada, 100% dos problemas estavam exatamente no nível calculado. Na híbrida sem LLM, 96,94% estavam no nível atual, e os demais estavam 100 pontos acima. Na híbrida com LLM, 99,29% estavam no nível atual. Em todas as abordagens, 100% respeitaram a faixa completa permitida.

Mesmo entre os problemas que não coincidiram com o conjunto de avaliação, nenhum ficou fora da faixa. Na abordagem estruturada sem LLM, os 839 problemas sem coincidência estavam exatamente no nível calculado; na estruturada com LLM, ocorreu o mesmo com os 837 problemas. Na híbrida sem LLM, dos 777 problemas sem coincidência, 752 estavam no nível exato e 25 estavam 100 pontos acima. Na híbrida com LLM, dos 782 problemas sem coincidência, 777 estavam no nível exato e 5 estavam 100 pontos acima.

Portanto, a ausência de coincidência com o futuro observado não significa automaticamente que o problema estava fora do nível do usuário. Ela significa apenas que aquele identificador não apareceu no conjunto de avaliação, embora ainda respeitasse as regras de *tag*, dificuldade e do passado observado.

4.1.5 Indicador de Avanço por *Tag*

A amostra contém 255 combinações entre usuário e *tag*, calculadas por:

$$85 \text{ usuários} \times 3 \text{ } tags = 255 \text{ combinações.} \tag{19}$$

Para cada combinação, foi considerado o nível calculado usando o passado observado. Em seguida, somou-se a quantidade de problemas resolvidos nesse mesmo nível no passado com a quantidade encontrada no futuro observado. O valor foi considerado atingido quando:

$$R_{passado}(u, t, n) + R_{futuro}(u, t, n) \geq 6, \tag{20}$$

em que u representa o usuário, t a *tag*, n o nível analisado e R a quantidade de problemas distintos resolvidos naquela combinação de *tag* e *rating*.

Das 255 combinações, 167 atingiram o limiar. O percentual foi calculado por:

$$\frac{167}{255} \times 100 = 65,49\%. \tag{21}$$

Essas 167 ocorrências estavam distribuídas entre 68 usuários distintos. Portanto, 68 dos 85 usuários, ou 80%, atingiram o limiar em pelo menos uma de suas três *tags*:

$$\frac{68}{85} \times 100 = 80\%. \tag{22}$$

Esse cálculo mostra onde a regra criada no sistema indicaria uma mudança de nível no futuro observado. Por exemplo, se um usuário possuía nível 900 em uma *tag* e a soma dos problemas do passado e do futuro observados nesse nível alcançou seis, seu indicador operacional passaria para 1000. O resultado descreve a aplicação da regra sobre os registros disponíveis.

4.1.6 Diversidade Semântica

A diversidade semântica média foi de 0,2552 na abordagem estruturada sem LLM, 0,2554 na estruturada com LLM, 0,2121 na híbrida sem LLM e 0,2169 na híbrida com LLM.

Em termos de variedade global, a quantidade de problemas distintos passou de 125 para 149 na recuperação estruturada e de 207 para 221 na recuperação híbrida quando a LLM foi utilizada.

Na comparação entre as abordagens híbridas, a inclusão da LLM aumentou a ILD de 0,2121 para 0,2169, uma diferença de 0,0048. Também houve aumento na variedade global: entre as 850 recomendações de cada abordagem, a híbrida sem LLM utilizou 207 problemas distintos, enquanto a híbrida com LLM utilizou 221. Isso representa 14 problemas distintos adicionais, ou um aumento de 6,76%.

Os resultados mostram comportamentos complementares entre as duas abordagens híbridas. A abordagem sem LLM apresentou 73 coincidências exatas, enquanto a abordagem com LLM manteve o mesmo *Hit Rate@10* de 44,71% e ampliou a diversidade semântica e a variedade global, utilizando 221 problemas distintos. Portanto, a LLM modificou a composição das listas e distribuiu as recomendações por um conjunto maior de problemas, sem reduzir a quantidade de usuários alcançados.

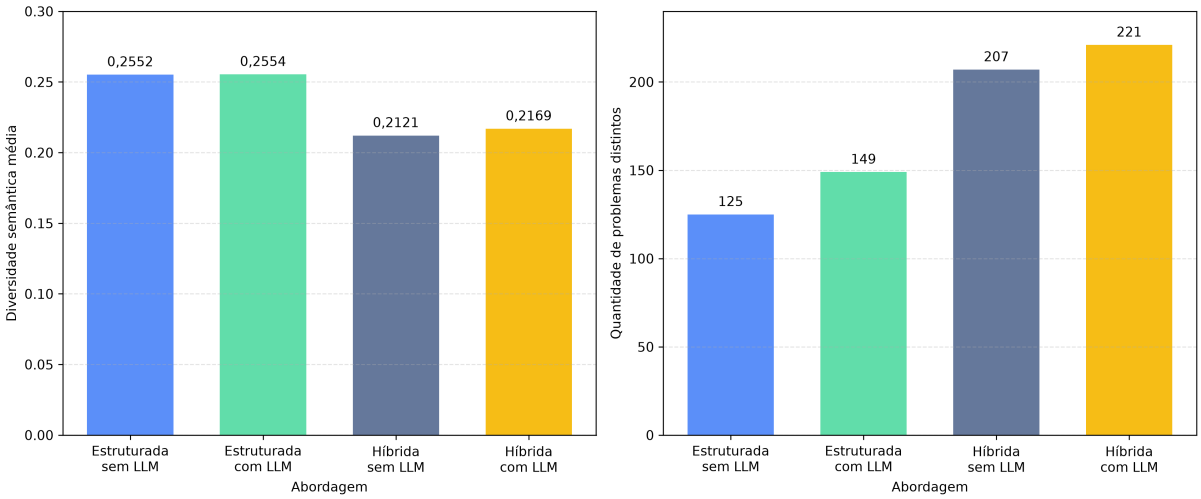

Figura 10 – Diversidade semântica e variedade global das quatro abordagens - Fonte: Autor

4.2 Análise e Discussão dos Resultados

Nesta seção os resultados quantitativos apresentados anteriormente são interpretados sob a visão das três hipóteses de pesquisa. A discussão avalia o impacto isolado do procedimento híbrido na correspondência de problemas, a influência da curadoria com a LLM sobre a diversidade das recomendações e a eficácia das regras de restrição na manutenção do controle da dificuldade.

4.2.1 Efeito do Procedimento Híbrido

Os resultados apoiam H1 para o procedimento híbrido completo. Sem LLM, o *Hit Rate@10* passou de 9,41% na recuperação estruturada para 44,71% na híbrida. Com LLM, passou de 10,59% para 44,71%. A Precisão@10 apresentou comportamento semelhante.

O ganho observado corresponde ao procedimento híbrido como implementado, que parte de até 50 problemas elegíveis por *tag* e utiliza a ordenação vetorial para manter até 15. Sob os parâmetros adotados, essa combinação apresentou maior correspondência com o futuro observado do que a recuperação estruturada.

4.2.2 Efeito da LLM

A H2 foi apoiada, no cenário avaliado, em relação à diversidade. Nas duas formas de recuperação, a LLM aumentou a quantidade de problemas distintos recomendados; na abordagem híbrida, também houve aumento da ILD.

A LLM foi tratada como um componente experimental de seleção. Seu uso forma a etapa generativa do RAG completo e permite reorganizar os problemas recuperados segundo o perfil apresentado no *prompt*. Os resultados mostram que essa reorganização

manteve o *Hit Rate@10* em 44,71% e ampliou a diversidade semântica e a variedade global das recomendações.

Como um problema que não aparece no futuro observado ainda pode respeitar a *tag* e a dificuldade do usuário, a diferença de acertos não transforma automaticamente as escolhas adicionais da LLM em recomendações inadequadas. Neste trabalho, o resultado é interpretado como uma diferença de comportamento entre as abordagens: a seleção sem LLM apresentou maior coincidência exata, enquanto a seleção com LLM apresentou maior diversificação.

Esse comportamento indica que a LLM atuou com sucesso como um mecanismo de reordenação (*re-ranker*) inteligente. Enquanto a busca vetorial tradicional aproxima textos puramente pela distância de cosseno, a LLM foi capaz de interpretar o contexto completo do *prompt*, que exigia a distribuição das recomendações entre diferentes assuntos e a priorização das fraquezas do usuário. Como resultado, a LLM evitou selecionar problemas excessivamente similares entre si. Ele explorou uma gama mais ampla de opções dentro do espaço de problemas elegíveis. Isso explica o aumento significativo na variedade global de identificadores únicos utilizados pelo sistema, comprovando que a curadoria generativa é capaz de enriquecer a diversidade das listas de estudo sem prejudicar a assertividade das recomendações.

4.2.3 Controle da Dificuldade

Os resultados apoiam H3 no sentido operacional. Todas as recomendações, inclusive aquelas que não coincidiram com o conjunto de avaliação, permaneceram dentro da faixa definida para a respectiva *tag*. Isso demonstra que a busca vetorial baseada em *embeddings* e a LLM atuaram somente depois da aplicação das regras de elegibilidade e não eliminaram o controle de dificuldade.

5 Conclusão

Este capítulo apresenta as conclusões obtidas a partir da pesquisa, retoma os objetivos definidos na introdução e destaca as principais contribuições do trabalho. Também são indicadas possibilidades de continuidade para o sistema e para novos experimentos.

O objetivo geral de desenvolver e avaliar um sistema híbrido de recomendação de problemas do Codeforces foi atingido. A pesquisa resultou em um sistema capaz de representar separadamente a proficiência do usuário em diferentes *tags*, selecionar problemas dentro de uma faixa de dificuldade e comparar abordagens estruturadas e híbridas, com e sem a participação de uma LLM.

Em relação à primeira hipótese (H1), os resultados sustentaram que o procedimento híbrido apresenta maior correspondência com o conjunto de avaliação (futuro observado) do que a recuperação exclusivamente estruturada. As duas abordagens híbridas alcançaram uma quantidade maior de usuários com pelo menos um acerto e produziram mais coincidências exatas com o futuro observado. Assim, a combinação dos filtros estruturados com a busca vetorial baseada em *embeddings* mostrou-se mais eficaz para recuperar problemas compatíveis com o comportamento posteriormente registrado.

A segunda hipótese (H2) também foi apoiada no cenário avaliado. A utilização da LLM ampliou a quantidade de problemas distintos presentes no conjunto total de recomendações e, na abordagem híbrida, também produziu um pequeno aumento da diversidade semântica média das listas. A LLM manteve a mesma quantidade de usuários com pelo menos uma coincidência na abordagem híbrida, embora a quantidade total de acertos tenha passado de 73 para 68. Assim, seu principal efeito observado foi modificar e ampliar a composição global das listas.

A terceira hipótese (H3) foi sustentada pelo controle da dificuldade. Todas as recomendações permaneceram entre o nível operacional calculado para cada *tag* e esse nível acrescido de 100 pontos. Isso ocorreu inclusive nas abordagens que utilizaram busca vetorial baseada em *embeddings* e seleção pela LLM, mostrando que esses componentes modificaram a seleção dos problemas sem eliminar as regras de dificuldade definidas pelo sistema.

Assim, a pesquisa chegou a três conclusões principais no cenário avaliado: o procedimento híbrido aumentou a correspondência com o futuro observado; a seleção por LLM esteve associada a uma maior variedade global e a um pequeno aumento da diversidade semântica; e os filtros mantiveram as recomendações dentro da dificuldade definida para cada *tag*. Esses resultados mostram que, para a amostra e os parâmetros adotados, a combinação avaliada é adequada selecionar problemas do Codeforces de forma personalizada e controlada.

5.1 Contribuições do Trabalho

As principais contribuições desta pesquisa são:

- ❏ a construção de um perfil de proficiência separado por *tag*, em vez de utilizar somente o *rating* geral do usuário;
- ❏ a ordenação das três *tags* selecionadas da menor para a maior proficiência estimada;
- ❏ a combinação de filtros de assunto, dificuldade e problemas ainda não resolvidos com recuperação semântica por busca vetorial baseada em *embeddings*;
- ❏ a implementação de um fluxo de RAG no qual a LLM recebe somente problemas recuperados e tem sua saída limitada aos identificadores fornecidos;
- ❏ a comparação das quatro abordagens sobre os mesmos 85 usuários, utilizando o mesmo histórico de construção (passado) e o mesmo conjunto de avaliação (futuro);
- ❏ a apresentação de resultados absolutos e percentuais para *Hit Rate@10*, Precisão@10 e MAE de *rating*;
- ❏ a identificação de que o procedimento híbrido trouxe ganho nas métricas de acerto e de que a LLM manteve o *Hit Rate@10* da abordagem híbrida, ao mesmo tempo que ampliou a diversidade semântica e a variedade global das recomendações.

5.2 Limitações da Pesquisa

Embora a pesquisa tenha atingido seus objetivos, algumas limitações metodológicas e operacionais devem ser destacadas:

1. Avaliação *Offline*: A validação do sistema baseou-se em dados históricos (futuro observado). Embora seja um protocolo padrão e rigoroso, ele não afere a percepção real, a satisfação ou a evolução efetiva de aprendizado que o competidor teria ao interagir ativamente com as recomendações.
2. Restrições da Amostra e do Perfil: A amostra limitou-se a 85 usuários iniciantes e intermediários (*rating* 600 a 1400). Comportamentos de usuários de níveis avançados ou profissionais podem diferir. Além disso, o perfil multitag considerou apenas os 3 assuntos mais frequentes de cada usuário, podendo negligenciar áreas de interesse secundárias nas quais o estudante também precisasse melhorar.
3. Limitações da LLM: O uso de LLM na etapa de geração adiciona latência ao sistema e apresenta custos computacionais restritivos (limite de *tokens*), o que motivou a redução da amostra e poderia ser um gargalo em um ambiente de produção em tempo real.

5.3 Trabalhos Futuros

Como trabalho futuro, recomenda-se repetir o experimento com uma amostra maior e incluir usuários da categoria *Specialist*. A repetição com diferentes amostras ajudará a verificar se os resultados permanecem semelhantes para outros perfis.

Outro experimento poderá igualar a quantidade inicial de problemas das recuperações estruturada e híbrida. As duas abordagens poderão partir do mesmo conjunto, mantendo a ordenação original em uma delas e aplicando a busca vetorial baseada em *embeddings* na outra. Essa comparação permitirá analisar de forma mais específica o efeito da ordenação semântica.

A regra de seis problemas por nível poderá ser submetida a uma análise de sensibilidade. A comparação entre diferentes valores permitirá observar como esse parâmetro modifica os níveis calculados e as recomendações. Também poderão ser avaliadas outras formas de estimar a proficiência por assunto.

O sistema pode ser ampliado para permitir que o competidor escolha entre aprofundar as *tags* mais praticadas e explorar assuntos menos frequentes. Essa opção atenderia os competidores que já resolveram muitos problemas do mesmo assunto e desejam mudar o foco de estudo. Nesse contexto, poderá ser desenvolvida uma interface para apresentar o nível de cada *tag*, o progresso até o próximo nível e a justificativa de cada recomendação. Essa visualização poderá tornar o perfil e as escolhas do sistema mais claros para o competidor.

Novos experimentos podem comparar modelos de *embeddings*, formas diferentes de construir a consulta semântica e diferentes LLMs. Além das métricas de acerto, podem ser medidos custo, quantidade de *tokens*, tempo de resposta e estabilidade das escolhas.

A coleta também pode registrar separadamente todas as tentativas, incluindo respostas incorretas, e o primeiro aceite de cada problema. Esses dados permitiriam estudar o esforço empregado na resolução sem contar várias submissões aceitas do mesmo problema como exercícios diferentes.

Repetições do experimento e análises estatísticas pareadas poderão complementar os resultados descritivos apresentados neste trabalho.

Referências

ANSARI, M. et al. Coders: A hybrid recommender system for an e-learning system. In: **2016 International Conference on Signal Processing and Intelligent Systems (ICSPIS)**. [S.l.]: IEEE, 2016. Citado 3 vezes nas páginas 30, 31 e 32.

CHIA, E. J.; NAJAFABADI, M. K. Solving cold start problem for recommendation system using content-based filtering. In: **Proceedings - 2022 International Conference on Computer Technologies, ICCTech 2022**. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2022. p. 38–42. ISBN 9781665499187. Citado na página 22.

Codeforces. **Codeforces API: Methods and Return Objects**. 2026. Acesso em: 23 jul. 2026. Disponível em: <https://codeforces.com/apiHelp?locale=en>. Citado 5 vezes nas páginas 7, 18, 19, 20 e 33.

DINU, I. G. **Codeforces Competitive Programming Dataset**. 2025. Kaggle. Acesso em: 29 jul. 2026. Disponível em: <https://www.kaggle.com/datasets/dinuiongeorge/codeforces-competitive-programming-dataset>. Citado na página 33.

DINU, I. G.; MIHAESCU, C.; REBEDEA, T. Matching problem statements to editorials in competitive programming. In: **2024 IEEE International Conference on Advanced Learning Technologies (ICALT)**. [S.l.]: IEEE, 2024. p. 171–175. Citado na página 33.

HOLTZMAN, A. et al. The curious case of neural text degeneration. In: **International Conference on Learning Representations**. Adis Abeba: OpenReview.net, 2020. Disponível em: <https://openreview.net/forum?id=rygGQyrFvH>. Citado na página 25.

JULIANTI, M. R. et al. Recommendation system model for personalized learning in higher education using content-based filtering method. In: **Proceedings of 2022 International Conference on Information Management and Technology, ICIMTech 2022**. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2022. p. 1–6. ISBN 9781665450904. Citado 2 vezes nas páginas 14 e 22.

Kaggle. **Kaggle: Your Machine Learning and Data Science Community**. 2026. <https://www.kaggle.com>. Acesso em: 29 jul. 2026. Citado 3 vezes nas páginas 7, 20 e 21.

LEE, C.-R.; LAO, H. Recommendation system for online programming judge platforms. In: **2019 IEEE 5th International Conference on Big Data Intelligence and Computing (DATACOM)**. [S.l.]: IEEE, 2019. p. 158–165. ISBN 978-1-7281-4117-6. Citado 7 vezes nas páginas 13, 17, 18, 22, 29, 30 e 32.

LEWIS, P. et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. **Advances in Neural Information Processing Systems**, v. 33, p. 9459–9474, 2020. Citado 5 vezes nas páginas 14, 23, 24, 25 e 31.

LIM, C. Y. Competitive learning: A paradigm for excellence, anchored in competitive programming. **The Journal of Competitive Learning**, v. 2, n. 1, Jul. 2026. Disponível em: <https://journals.u.icpc.global/jcl/article/view/7>. Citado na página 13.

MALKOV, Y. A.; YASHUNIN, D. A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 42, n. 4, p. 824–836, 2020. Citado na página 24.

MUEPU, D. M.; WATANOBE, Y.; AMIN, M. F. I. A comprehensive content-based recommendation system for programming problems through multi-faceted code analysis. **IEEE Access**, v. 13, p. 93712–93734, 2025. ISSN 2169-3536. Citado 11 vezes nas páginas 14, 17, 18, 22, 23, 26, 27, 28, 30, 31 e 32.

PAPERT, S. **Mindstorms: Children, Computers, and Powerful Ideas**. New York: Basic Books, 1980. ISBN 0-465-04627-4. Citado 2 vezes nas páginas 14 e 21.

PGVECTOR. **pgvector: Open-source vector similarity search for Postgres**. 2026. Acesso em: 27 jul. 2026. Disponível em: <https://github.com/pgvector/pgvector>. Citado na página 24.

RAMESH, P. N.; KANNIMUTHU, S. Context-aware practice problem recommendation using learners' skill level navigation patterns. **Intelligent Automation & Soft Computing**, v. 35, n. 3, p. 3843–3857, 2023. Citado 4 vezes nas páginas 13, 30, 31 e 32.

RICCI, F.; ROKACH, L.; SHAPIRA, B. **Recommender Systems Handbook**. 2nd. ed. [S.l.]: Springer, 2015. Citado 5 vezes nas páginas 14, 23, 24, 26 e 27.

SHUTE, V. J. Stealth assessment in computer-based games to support learning. In: TOBIAS, S.; FLETCHER, J. D. (Ed.). **Computer Games and Instruction**. Charlotte, NC: Information Age Publishing, 2011. p. 503–523. Disponível em: <https://myweb.fsu.edu/vshute/pdf/shute%20pres_h.pdf>. Citado 2 vezes nas páginas 14 e 22.

SMYTH, B.; MCCLAVE, P. Similarity vs. diversity. In: AHA, D. W.; WATSON, I. (Ed.). **Case-Based Reasoning Research and Development**. Berlin, Heidelberg: Springer, 2001. (Lecture Notes in Computer Science, v. 2080), p. 347–361. Citado na página 28.

TOLEDO, R. Y.; MOTA, Y. C.; MARTíNEZ, L. A recommender system for programming online judges using fuzzy information modeling. **Informatics**, v. 5, 2018. ISSN 22279709. Citado 7 vezes nas páginas 13, 17, 18, 22, 29, 30 e 32.

VYGOTSKY, L. S. **Mind in Society: The Development of Higher Psychological Processes**. Cambridge, MA: Harvard University Press, 1978. ISBN 978-0-674-57629-2. Citado 2 vezes nas páginas 14 e 21.

YOSHIMURA, R. et al. Wojr: A recommendation system for providing similar problems to programming assignments. **Applied System Innovation**, v. 5, p. 53, 5 2022. ISSN 2571-5577. Citado 6 vezes nas páginas 13, 17, 18, 30, 31 e 32.

ZAFFALON, F. et al. A recommender system of computer programming exercises based on student's multiple abilities and skills model. In: **2022 IEEE Frontiers in Education Conference (FIE)**. [S.l.]: IEEE, 2022. p. 1–8. ISBN 978-1-6654-6244-0. Citado 5 vezes nas páginas 13, 18, 30, 31 e 32.

Apêndices

APÊNDICE A – Trechos de Código e Prompts

Este apêndice apresenta os trechos principais usados para reproduzir as regras e as métricas descritas no trabalho. Os nomes internos de algumas classes ainda utilizam a palavra *candidate*; nesse contexto, o termo representa um problema recuperado antes da composição da lista final, e não um usuário.

A.1 Remoção de repetições e seleção das três tags

Um mesmo usuário pode enviar e obter *Accepted* mais de uma vez para o mesmo problema, inclusive com códigos ou linguagens diferentes. Para evitar que essas repetições aumentem artificialmente a frequência das *tags*, os problemas são agrupados pelo identificador oficial do Codeforces. Em seguida, contam-se as *tags* dos problemas distintos e selecionam-se as três mais frequentes.

Listing A.1 – Remoção de problemas repetidos e seleção das tags

```
private List<Problem> uniqueProblems(List<Problem> problems) {
    return new ArrayList<>(problems.stream()
        .filter(problem -> problem.getCfProblemId() != null)
        .collect(Collectors.toMap(
            Problem::getCfProblemId,
            Function.identity(),
            (first, ignored) -> first,
            LinkedHashMap::new))
        .values());
}

public List<String> selectTopTags(
        List<Problem> trainingProblems, int numberOfTags) {
    return countTagFrequencies(uniqueProblems(trainingProblems))
        .entrySet().stream()
        .sorted(Map.Entry.<String, Integer>comparingByValue()
            .reversed().thenComparing(Map.Entry::getKey))
        .limit(numberOfTags)
        .map(Map.Entry::getKey)
        .toList();
}
```

A.2 Cálculo do nível operacional por tag

O nível começa em 800 e avança em intervalos de 100 enquanto existirem pelo menos seis problemas distintos resolvidos naquela *tag* e naquele *rating*. O número seis é um parâmetro operacional definido neste trabalho para calcular a progressão do nível por tag.

Listing A.2 – Regra de progressão por tag

```
Map<Integer, Long> solvedByRating = trainingProblems.stream()
    .filter(problem -> problem.getRating() != null)
    .filter(problem -> normalizedTags(problem).contains(tag))
    .collect(Collectors.groupingBy(
        Problem::getRating, Collectors.counting()));

int currentLevel = baseRating;
while (solvedByRating.getOrDefault(currentLevel, 0L)
        >= requiredSolvesPerLevel) {
    currentLevel += ratingStep;
}
```

Na execução avaliada, os parâmetros foram `baseRating = 800`, `ratingStep = 100` e `requiredSolvesPerLevel = 6`.

A.3 Recuperação híbrida e busca vetorial

Primeiro, são recuperados até 50 problemas por *tag* que respeitam a faixa de *rating* e que ainda não foram resolvidos pelo usuário. A consulta textual do perfil é convertida em *embedding* pelo mesmo modelo usado na indexação. O PGVector compara esse vetor com os vetores armazenados e devolve os problemas semanticamente mais próximos. Somente problemas presentes no conjunto estruturado podem permanecer no resultado.

Listing A.3 – Consulta ao banco vetorial

```
String filter = "tag == '" + escapeFilter(proficiency.tag())
    + "' && rating >= " + proficiency.currentLevel()
    + " && rating <= "
    + (proficiency.currentLevel() + ratingStep);

SearchRequest request = SearchRequest.builder()
    .query(buildSemanticQuery(proficiency, recentTrainingProblems))
    .topK(Math.max(semanticLimit * 8, structuredLimit))
    .similarityThresholdAll()
    .filterExpression(filter)
```

```
    .build();

for (Document document : vectorStore.similaritySearch(request)) {
    String id = String.valueOf(
        document.getMetadata().get("cfProblemId"));
    Problem problem = eligible.get(id);
    if (problem != null && added.add(id)) {
        ranked.add(new RankedProblemCandidate(
            problem,
            proficiency.tag(),
            priority,
            structuredRank.get(id),
            document.getScore(),
            RetrievalMode.HYBRID));
        if (ranked.size() == semanticLimit) break;
    }
}
```

No trecho, `structuredLimit` vale 50 e `semanticLimit` vale 15. Portanto, 50 é o limite inicial por *tag*, 15 é o limite mantido por *tag* após a recuperação e 10 é o tamanho da lista final. Quando existem três *tags*, a fusão pode conter até 45 problemas antes da curadoria.

A.4 Geração dos documentos do PGVector

O índice cria um documento para cada combinação válida entre problema e *tag*. Por isso, os 20.634 documentos registrados na execução não significam 20.634 problemas únicos. Um problema com três *tags* produz três documentos, permitindo aplicar o filtro da *tag* durante a consulta. Cada documento recebe um identificador determinístico, conteúdo textual, metadados e um *embedding* com 768 dimensões.

Listing A.4 – Criação de um documento por problema e tag

```
for (String rawTag : problem.getTags()) {
    String tag = normalizeTag(rawTag);
    String documentId = deterministicDocumentId(
        problem.getCfProblemId(), tag);

    metadata.put("cfProblemId", problem.getCfProblemId());
    metadata.put("tag", tag);
    metadata.put("rating", problem.getRating());
    metadata.put("title", safe(problem.getTitle()));

    batch.add(new Document(
```

```
        documentId,
        buildEmbeddingText(problem, tag, includeStatement),
        metadata));
}
```

O valor gravado pode ser conferido diretamente no banco após a indexação:

Listing A.5 – Contagem dos documentos vetoriais

```
SELECT COUNT(*) AS documentos_vetoriais
FROM problem_vector_store_768;

SELECT COUNT(DISTINCT metadata->>'cfProblemId')
       AS problemas_distintos
FROM problem_vector_store_768;
```

A.5 Prompts usados pela LLM

O *prompt* de sistema foi construído para limitar a função da LLM à escolha dos identificadores dos problemas já recuperados. Seu conteúdo foi:

> Você é o curador final de um sistema RAG educacional. Selecione exatamente 10 problemas somente dentre os problemas fornecidos. Priorize as tags na ordem de menor para maior proficiência e respeite a dificuldade alvo. Distribua as recomendações entre todas as tags. Responda apenas com IDs oficiais separados por espaço, sem explicações e sem inventar IDs. Perfil: [perfil calculado do usuário].

O *prompt* do usuário contém, para no máximo 15 problemas, os campos: identificador oficial, título, *tag*-alvo, *rating*, pontuação semântica e conjunto de *tags*. O enunciado não é enviado à LLM. Essa escolha reduz o uso de tokens e mantém apenas as informações necessárias para a curadoria estudada.

A temperatura foi configurada como zero por meio da propriedade:

Listing A.6 – Configuração da temperatura

```
spring.ai.ollama.chat.options.temperature=0.0
```

A temperatura controla o grau de variação na escolha dos próximos elementos da resposta. O valor zero busca tornar a saída mais estável e reproduzível, mas não garante sozinho a ausência de respostas inválidas. Por esse motivo, o sistema extrai somente identificadores que pertencem à lista enviada e completa qualquer posição faltante com a ordenação determinística.

Listing A.7 – Validação dos identificadores devolvidos

```
Set<String> validIds = problems.stream()
    .map(problem -> problem.problem().getCfProblemId()
        .toUpperCase(Locale.ROOT))
    .collect(Collectors.toSet());

Arrays.stream(response.toUpperCase(Locale.ROOT)
        .replaceAll("[^A-Z0-9\\s]", " ")
        .split("\\s+"))
    .filter(validIds::contains)
    .forEach(extracted::add);
```

A.6 Cálculo das métricas

Para cada usuário, um acerto ocorre quando um identificador recomendado também aparece no conjunto de teste. A Precisão@10 é a quantidade de acertos dividida por dez. O *Hit Rate*@10 recebe valor um quando existe ao menos um acerto na lista e zero caso contrário. O resultado geral é a média entre os 85 usuários.

O MAE de *rating* usado neste trabalho é uma adaptação: calcula-se a diferença absoluta entre a média de dificuldade da lista recomendada e a média de dificuldade do conjunto de teste de cada usuário.

Listing A.8 – Cálculo de acertos, Precisão@10 e MAE de rating

```
Set<String> futureIdSet = new HashSet<>(testIds);
List<String> suggestedIds = selected.stream()
    .map(p -> p.problem().getCfProblemId()).toList();
List<String> hitIds = suggestedIds.stream()
    .filter(futureIdSet::contains).toList();

double meanRecommendedRating = selected.stream()
    .map(RankedProblemCandidate::problem)
    .map(Problem::getRating)
    .filter(Objects::nonNull)
    .mapToInt(Integer::intValue)
    .average().orElse(Double.NaN);

double meanFutureRating = futureProblems.stream()
    .map(Problem::getRating)
    .filter(Objects::nonNull)
    .mapToInt(Integer::intValue)
    .average().orElse(Double.NaN);
```

```
double precisionAt10 = (double) hitIds.size() / 10;
int userWithHit = hitIds.isEmpty() ? 0 : 1;
double ratingMae = Math.abs(
    meanRecommendedRating - meanFutureRating);
```

A.7 Cálculo das estatísticas da base

O script de auditoria soma os registros *Accepted*, os problemas distintos e as partes usadas para perfil e referência. Ele também calcula média, mínimo e máximo por usuário.

Listing A.9 – Estatísticas descritivas da amostra

```
def describe(values):
    return {
        "total": sum(values),
        "media": statistics.fmean(values),
        "minimo": min(values),
        "maximo": max(values),
    }

accepted = [int(raw_by_handle[row["Handle"]]["Total_ACs"])
           for row in final_rows]
distinct = [int(row["Total_Problemas_Distintos"])
           for row in final_rows]
training = [int(row["Treino_Problemas_Distintos"])
           for row in final_rows]
reference = [int(row["Teste_Problemas_Distintos"])
            for row in final_rows]

profile_fraction = sum(training) / sum(distinct)
repeated_accepted = sum(accepted) - sum(distinct)
```

Para formalizar a escolha das três *tags*, o script conta as *tags* distintas de cada usuário e mede quantos problemas possuem pelo menos uma das *tags* Top-K:

Listing A.10 – Média de tags e cobertura das três mais frequentes

```
frequencies = Counter()
for tags in mapped.values():
    frequencies.update(tags)

ranked_tags = sorted(
    frequencies,
```

```
    key=lambda tag: (-frequencies[tag], tag))
distinct_counts.append(len(frequencies))

selected = set(ranked_tags[:3])
covered = sum(
    1 for tags in mapped.values()
    if tags.intersection(selected))
coverage_top_3 = covered / len(mapped) if mapped else 0.0
union_coverages[3].append(coverage_top_3)

mean_distinct_tags = statistics.fmean(distinct_counts)
mean_top_3_coverage = statistics.fmean(union_coverages[3])
```

A cobertura é calculada separadamente para cada usuário antes da média. O resultado obtido foi de 17,32 *tags* distintas por usuário, enquanto as três mais frequentes cobriram 87,33% dos problemas mapeados, em média.

Com os dados finais, foram obtidos 29.289 registros *Accepted*, 27.273 pares distintos entre usuário e problema, 8.218 problemas no conjunto de treino e 19.055 no conjunto de teste. Assim,

$$\frac{8\,218}{27\,273} \times 100 = 30,1324\% \approx 30,13\%.$$