Spotify Recs
Sistema multiagentes para recomendação de músicas e gestão de playlist

Rhogger Freitas Silva, Pedro Henrique Mendes

***Abstract*—Este artigo apresenta o desenvolvimento de um sistema inteligente para recomendação e gestão musical integrado à plataforma Spotify. O projeto utiliza uma arquitetura baseada em sistemas multiagentes coordenados por Modelos de Linguagem de Grande Porte (LLMs), integrados a ferramentas externas através do Model Context Protocol (MCP). O sistema combina um motor de recomendação personalizado baseado no algoritmo K-Nearest Neighbors (KNN), treinado sobre um dataset de 160.000 faixas, com agentes autônomos capazes de gerenciar playlists, buscar músicas e controlar a reprodução em tempo real. A solução proposta demonstra uma abordagem inovadora para a descoberta musical, transformando a interação passiva em uma experiência conversacional e proativa.**

***Index Terms*—Autonomous Agents, Large Language Models (LLMs), Multi-Agent Systems, Natural Language Processing, Music Recommendation Systems, Intelligent Personal Assistants, Content-Based Music Discovery, Agentic Workflows**

I. Introdução

A indústria da música digital passou por uma transformação radical com o surgimento das plataformas de streaming, como o Spotify. Atualmente, os usuários têm acesso a catálogos contendo dezenas de milhões de faixas, o que, embora positivo, cria o desafio da descoberta de conteúdo relevante. Sistemas de recomendação tradicionais operam em segundo plano, muitas vezes de forma opaca, sugerindo músicas com base em algoritmos de filtragem colaborativa ou baseada em conteúdo.

No entanto, a interação com esses sistemas costuma ser limitada a interfaces gráficas passivas. Este trabalho apresenta o *Spotify Recs*, uma plataforma que utiliza Grandes Modelos de Linguagem (LLMs) e Sistemas Multiagentes (MAS) para criar um assistente pessoal inteligente. O sistema não apenas recomenda músicas usando um modelo de Machine Learning personalizado, mas também gerencia bibliotecas, cria playlists e controla a reprodução através de uma interface conversacional e fluxos agenticos autônomos.

II. Problema

Apesar do vasto catálogo disponível, muitos usuários enfrentam a ”paralisia de escolha” ou a dificuldade de expressar verbalmente o que desejam ouvir em termos de características técnicas (como ”danceabilidade” ou ”instrumentalidade”). Além disso, a gestão manual de playlists e a busca por músicas que se encaixem em contextos específicos podem ser tarefas onerosas.

Os sistemas atuais carecem de uma camada de inteligência que compreenda intenções complexas e execute sequências de ações de forma autônoma. Por exemplo, um usuário pode desejar ”encontrar músicas similares à ’Cure’ da banda Metallica e adicioná-las a uma nova playlist chamada ’Heavy Metal’”. Realizar isso hoje exige múltiplas interações manuais. O problema central, portanto, é a falta de uma interface de controle inteligente e unificada que combine recomendação técnica com execução operacional.

III. Arquitetura do sistema

A arquitetura do *Spotify Recs* é baseada em microserviços, garantindo escalabilidade e modularidade. O sistema é composto por quatro componentes principais:

- **Frontend:** Uma interface web moderna desenvolvida em Nuxt que fornece de forma fácil a gestão de playlists, recomendação manual através dos filtros (audio features e metadados), o chat e controles de player.
- **Backend (Core):** Desenvolvido com FastAPI, atua como o cérebro do sistema, hospedando a lógica dos agentes, integração com MCP e demais componentes.
- **MCP Server:** Um servidor baseado no *Model Context Protocol* que abstrai e fornece ferramentas de integração com a API do Spotify.
- **Projeto de ML:** Um projeto dedicado para processamento de dados e geração de modelos de recomendações usando o algoritmo K-Nearest Neighbors (KNN).

O fluxo de dados ocorre de forma orquestrada: o usuário envia um prompt ou executa alguma ação através de botões via Frontend, o Backend processa a intenção através de agentes especialistas que utilizam o MCP para interagir com o Spotify ou o serviço de ML para recomendações avançadas, e retorna a resposta final ao usuário.

IV. Infraestrutura

O *Spotify Recs* é implantado em uma infraestrutura conteinerizada para garantir isolamento e reprodutibilidade. A orquestração é realizada através do Docker Compose, definindo um ecossistema de microserviços que se comunicam através de uma rede isolada (`spotify_net`).

- **Isolamento de Redes:** Os serviços de banco de dados e o servidor MCP não são expostos diretamente para a internet, sendo acessíveis apenas pelo backend. O frontend comunica-se com o backend via porta pública 8000, enquanto a base de dados opera na porta padrão 5432 protegida por permissões internas do Docker.

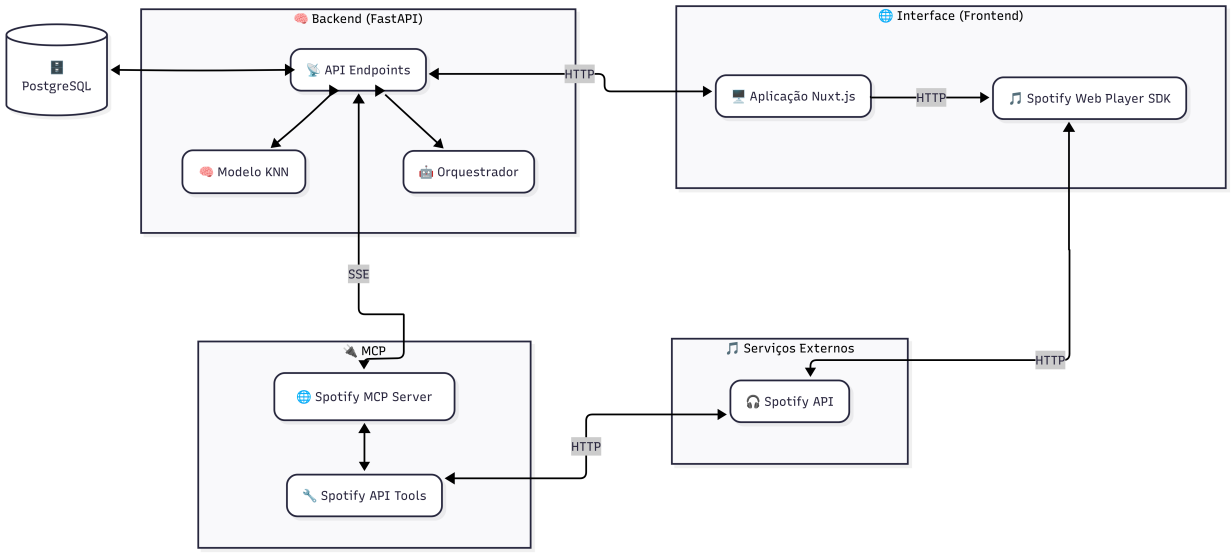

Fig. 1. Diagrama da Arquitetura.

- **Persistência de Dados:** Volumes nomeados (*Docker Volumes*) são utilizados para garantir que os dados do banco PostgreSQL e os metadados do servidor MCP persistam entre reinicializações dos contêineres.
- **Gerenciamento de Dependências:** Cada componente possui seu próprio `Dockerfile` otimizado: o backend utiliza `pipenv`, o frontend `pnpm` com Nuxt, e o servidor MCP `node.js`. Isso permite o escalonamento independente de cada parte conforme a carga do sistema.
- **Ambiente de Desenvolvimento:** A infraestrutura suporta *hot-reload* (montagem de volumes locais no contêiner), permitindo que alterações no código-fonte sejam refletidas instantaneamente sem necessidade de reconstrução da imagem.

V. Backend (API)

O backend do sistema é construído sobre o framework FastAPI, utilizando uma arquitetura modular que separa as preocupações entre API, serviços e orquestração de agentes. A escolha do FastAPI justifica-se pela sua tipagem estática via Pydantic, o que minimiza erros em tempo de execução e gera documentação automática (OpenAPI/Swagger) essencial para o desenvolvimento ágil.

- **Serviços e Reusabilidade:** Centraliza a lógica em serviços como `PlaylistsService`, que alimentam tanto endpoints REST quanto as *Tools* dos agentes, garantindo consistência operacional.
- **Contratos (Schemas):** Utiliza Pydantic para definir contratos rigorosos (ex: `TrackResponse`), assegurando a integridade dos dados entre o banco, agentes e frontend.
- **Operações Assíncronas:** Emprega `asyncio` para gerenciar o protocolo MCP (via SSE) e chamadas externas de alta latência concomitantes.
- **Segurança OAuth2:** Gerencia de forma automatizada o ciclo de vida de *tokens* do Spotify no PostgreSQL, garantindo acesso persistente e seguro.
- **Orquestração ADK:** Utiliza o Google ADK para reger o sistema multiagente, configurando instruções que definem o comportamento e as restrições dos modelos Gemini.

VI. Serviços Externos

O sistema integra-se ao ecossistema Spotify através de duas frentes:

- **Spotify Web API:** Utilizada para obter metadados, "Audio Features" técnicas, gestão de playlists e consulta de histórico.
- **Playback SDK:** Combina a *Player API* para controle remoto (via Spotify Connect) com o *Web Playback SDK* integrado ao frontend, permitindo que a aplicação atue como um reprodutor nativo com feedback em tempo real.

VII. Sistema Multiagentes

A inteligência do projeto reside em sua natureza multiagente. Em vez de um único prompt monolítico, o sistema utiliza uma equipe de especialistas coordenados por um orquestrador central. Esta abordagem aumenta a precisão e a robustez, permitindo que cada agente seja otimizado para uma tarefa específica.

A. Agentes

Na camada agêntica, cinco agentes principais foram implementados:

- **Orquestrador (Orchestrator):** Atua como o ponto de entrada e cérebro central. Analisa a intenção em linguagem natural, planeja a execução e coordena a comunicação entre os sub-agentes. Ele garante que a resposta final seja humanizada e natural, mantendo

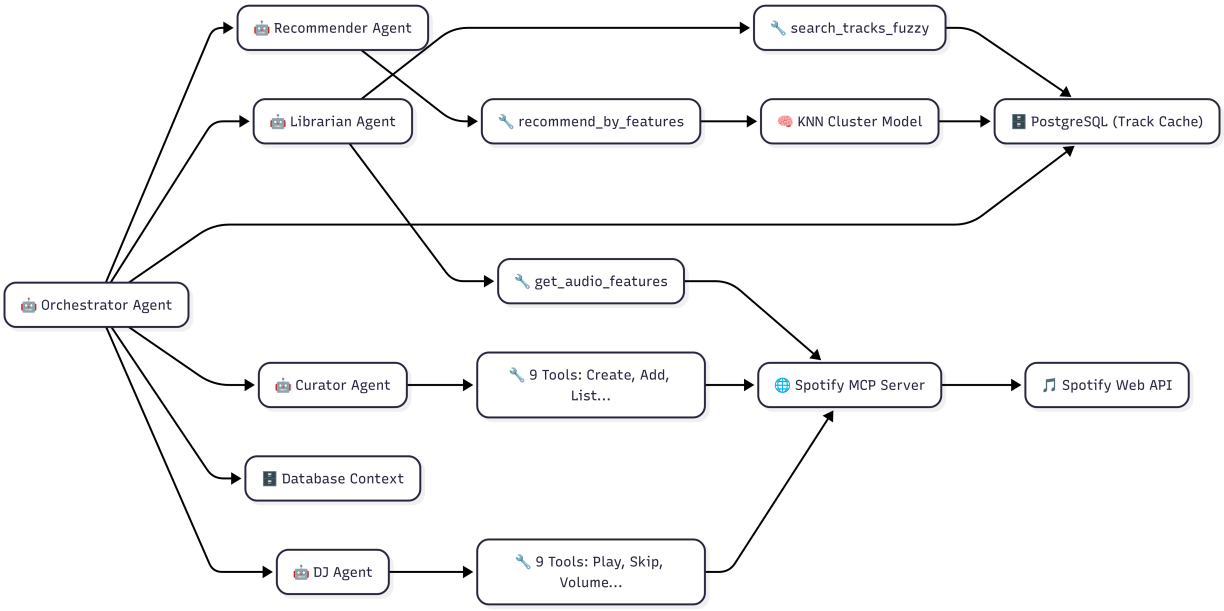

Fig. 2. Diagrama da camada agêntica.

uma persona unificada e ocultando a complexidade da colaboração interna.

- **Librarian (Especialista em Catálogo):** Responsável por localizar faixas, artistas e álbuns. Sua principal característica é a busca *fuzzy* no banco de dados local, permitindo que o sistema encontre músicas mesmo com entradas imprecisas ou com erros ortográficos. Ele fornece os metadados técnicos necessários para as ferramentas de recomendação.
- **Curator (Gestor de Biblioteca):** Especializado na organização da coleção pessoal do usuário. Ele gerencia o ciclo de vida das playlists (criação, edição e exclusão) e possui capacidades de busca de pertinência, sendo capaz de identificar em quais coleções específicas uma determinada música está presente.
- **DJ (Controlador de Reprodução):** Responsável pelo controle em tempo real do player. Ele traduz comandos do usuário para ações diretas como ajuste de volume, troca de faixas, controle de fila (*queue*) e modos de reprodução (*shuffle/repeat*), agindo sobre o dispositivo ativo.
- **Recommender (Especialista em Descoberta):** Foca na geração de sugestões baseadas em alguma música como contexto ou suas características técnicas. Ele consome o modelo de Machine Learning para encontrar similaridades por atributos de áudio, garantindo que as recomendações mantenham a coesão sonora solicitada pelo usuário.

Essa especialização permite que cada agente opere com instruções (*system instructions*) otimizadas, aumentando a precisão do sistema e facilitando a manutenção de fluxos complexos.

B. Tools

As capacidades de ação dos agentes são viabilizadas através de um conjunto de ferramentas (*tools*) especializadas, que abstraem a complexidade das APIs e do banco de dados. Estas ferramentas são agrupadas por domínios de competência:

- **Busca e Inteligência Local (Librarian & Recommender):** Este grupo foca na extração de dados e descoberta musical. Inclui ferramentas de busca *fuzzy* no banco de dados PostgreSQL para localização rápida de faixas e artistas, e ferramentas de inferência que consultam o modelo KNN para gerar recomendações baseadas no perfil técnico de áudio (*audio features*).
- **Gestão de Biblioteca (Curator):** Ferramentas voltadas à manipulação estruturada do ecossistema do usuário. Abrange desde a criação e edição de playlists até funções avançadas de inspeção, como a listagem de faixas de coleções específicas e a verificação de existência de uma música em múltiplas playlists. Estas ações utilizam majoritariamente as capacidades expostas pelo servidor MCP.
- **Controle de Reprodução (DJ):** Conjunto de ferramentas de baixa latência para interação direta com os dispositivos ativos. Permite o disparo de execuções musicais (via URI ou busca), manipulação do estado do player (*play/pause/skip*) e ajustes de configuração de ambiente, como controle de volume, modos de repetição e ordenação aleatória.

Esta separação por contextos garante que o Orquestrador possa delegar tarefas de forma pontual, permitindo que os agentes combinem múltiplas ferramentas para resolver solicitações complexas (ex: buscar músicas $\rightarrow$ recomendar similares $\rightarrow$ adicionar em nova playlist).

VIII. MCP

A utilização do *Model Context Protocol* (MCP) representa uma vantagem estratégica na arquitetura do sistema, atuando como uma camada de abstração padronizada que expõe capacidades do mundo real para o modelo de linguagem. O servidor MCP opera como um serviço independente, o que garante o desacoplamento entre a lógica de integração externa e a orquestração dos agentes. Esta separação facilita a escalabilidade horizontal, permitindo que novos servidores MCP sejam adicionados para suportar diferentes fontes de dados ou APIs sem a necessidade de modificar profundamente os agentes existentes, além de proteger o backend contra mudanças estruturais nas APIs externas.

Através do protocolo MCP, as ferramentas (*tools*) são autodocumentadas e descobertas dinamicamente pelo sistema via esquemas JSON-RPC. Funções como controle de reprodução e busca no catálogo são expostas com metadados claros sobre parâmetros e tipos de retorno, permitindo que o sistema evolua de forma orgânica. Este mecanismo de introspecção garante que novas funcionalidades adicionadas ao servidor MCP tornem-se instantaneamente utilizáveis pelos agentes, promovendo uma arquitetura flexível e de fácil manutenção.

IX. Modelo de Recomendação

O motor de recomendação baseia-se em uma abordagem de filtragem baseada em conteúdo, utilizando o algoritmo *K-Nearest Neighbors* (KNN) para encontrar faixas similares em um espaço multidimensional.

1) Análise Exploratória (EDA) e Seleção de Atributos: O desenvolvimento iniciou-se com a análise de um dataset bruto contendo 170.000 faixas. Durante a análise exploratória, observou-se através de matrizes de correlação que certas variáveis, apesar de tecnicamente relevantes, introduziam ruído à recomendação de "estilo musical". Atributos como `key`, `mode` e `loudness` apresentaram baixa correlação com a percepção subjetiva de similaridade sonora neste algoritmo.

Como resultado, foi realizada uma eliminação estratégica de atributos, reduzindo o espaço vetorial de 19 para 13 variáveis. Foram descartadas características como `liveness` (que indicava presença de público, mas não o estilo da composição) e `tempo`, permitindo que o modelo se concentrasse em variáveis densas como `valence` (positividade musical) e `danceability`. A EDA revelou ainda uma correlação negativa acentuada de -0,75 entre `acousticness` e `energy`, validando a integridade do dataset para a distinção de gêneros e atmosferas.

2) Engenharia de Features e Pré-processamento: Para garantir que o modelo capture tanto a sonoridade quanto o contexto histórico, foi aplicada uma pipeline composta por:

- **Normalização:** Atributos como `acousticness`, `danceability`, `energy` e `valence` foram padronizados via *StandardScaler*, crucial para o cálculo preciso da Distância Euclidiana.
- **Binarização de Popularidade:** Utilizou-se a mediana de popularidade (33) como ponto de corte para a flag `is_popular`, equilibrando recomendações *mainstream* e de nicho.
- **Categorização Temporal:** Aplicou-se *One-Hot Encoding* em 11 décadas (1920s a 2020s), fixando a época da música como uma dimensão fundamental da vizinhança.

3) Arquitetura e Avaliação do Modelo: O modelo KNN utiliza o algoritmo **KD-Tree**, otimizado para o corpus de 169.907 registros. Nos testes de avaliação, o motor demonstrou alta eficiência com latência de **0,16ms** por recomendação. O sucesso da estratégia de seleção de features é evidenciado pela métrica de **Cauda Longa**, onde 50,19% das sugestões são de músicas não-populares, e pelo score de **Serendipidade** (0,0126), que confirma a capacidade do modelo de oferecer descobertas coesas porém não óbvias.

X. Banco de dados

O sistema utiliza PostgreSQL para persistência de dados, operado via SQLAlchemy. A estrutura foca no suporte à inteligência do sistema e na gestão de identidade:

- **Usuários e Sessão:** Gerencia o perfil do usuário e o ciclo de vida dos *tokens* OAuth2 do Spotify, garantindo autenticação persistente.
- **Catálogo Inteligente:** Armazena faixas musicais com seus respectivos vetores de *audio features* e décadas já processados, servindo de base de dados otimizada para as consultas de vizinhança do modelo KNN.
- **Interações e Feedback:** Registra comportamentos implícitos (como *skips*) e preferências explícitas (*likes*), permitindo correlacionar o perfil do usuário com as características técnicas das músicas consumidas.
- **Seeding Automatizado:** O sistema possui um mecanismo de *bootstrap* que, na primeira execução, popula automaticamente o PostgreSQL com o *corpus* musical processado via CSV, garantindo que o motor de recomendações esteja operacional imediatamente após o a disponibilidade do banco.

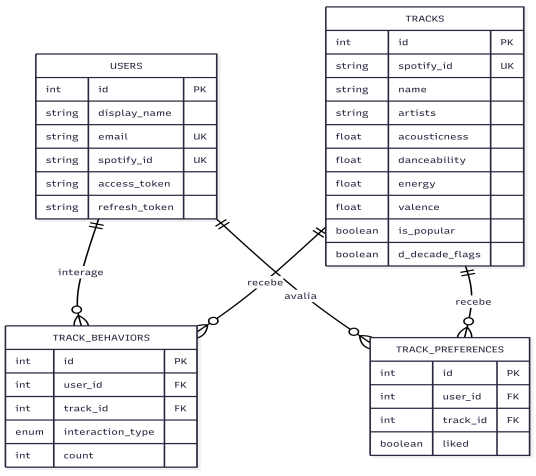

Fig. 3. Diagrama de Entidade Relacionamento.

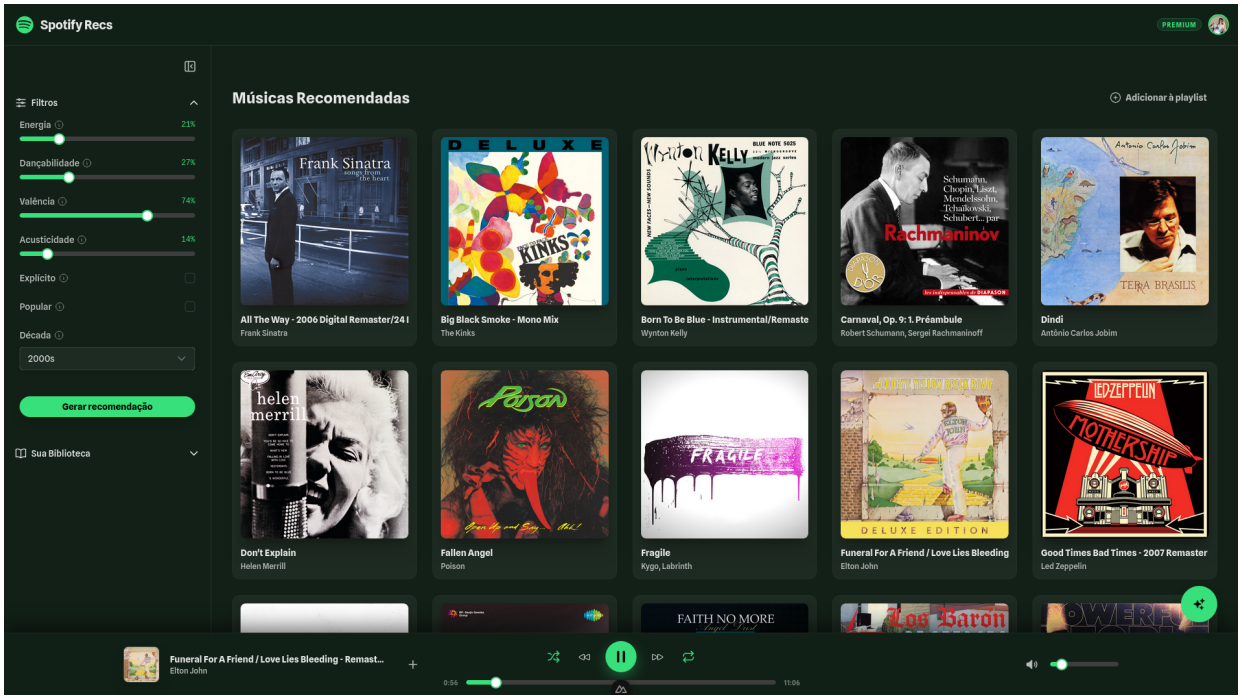

Fig. 4. Tela de recomendação musical.

XI. Frontend (Interface do Usuário)

A interface em Nuxt 4 e Nuxt UI provê uma experiência de chat reativa com suporte a componentes dinâmicos (*cards* interativos). A integração nativa com o *Spotify Web Playback SDK* permite o controle bidirecional em tempo real, transformando o navegador em um reprodutor autêntico sob o comando dos agentes e garantindo uma interação fluida entre o assistente e o player.

XII. Conclusão

O projeto *Spotify Recs* demonstra que a eficácia de um assistente musical moderno reside na simbiose entre a flexibilidade dos sistemas multiagentes (MAS) e a precisão analítica do Machine Learning. O trabalho validou que a inteligência do sistema não advém apenas de sua infraestrutura, mas da colaboração entre agentes especialistas e um motor de recomendação KNN rigorosamente calibrado, capazes de transformar a descoberta musical em um processo ativo e contextualizado.

A principal contribuição acadêmica e técnica desta implementação é a prova de que o uso de agentes autônomos para ”humanizar” o acesso a modelos de ML complexos resolve o problema da paralisia de escolha. Através da orquestração de fluxos, os agentes utilizam atributos técnicos de áudio para promover a exploração da ”cauda longa” do catálogo (50,19% das sugestões), garantindo que a precisão estatística do modelo se converta em valor real e natural para o usuário final em tempo de execução (0,16ms).

Em conclusão, o sistema posiciona-se como uma evolução sobre os modelos tradicionais de busca e recomendação, consolidando o uso de sistemas agênticos como o paradigma ideal para interfaces que busquem unir curadoria técnica profunda a uma experiência de uso intuitiva e fluida.

XIII. Trabalhos Futuros

O desenvolvimento do *Spotify Recs* apresenta um roteiro claro de evoluções planejadas para expandir a inteligência e a utilidade do sistema:

- **Gestão de Sessões de Chat:** Implementar a persistência de sessões, permitindo que o usuário retome diálogos anteriores e que os agentes mantenham um histórico contextual de longo prazo sobre as conversas realizadas.
- **Histórico e Cadastro de Filtros:** Criar um mecanismo para registro e descrição de filtros personalizados. Isso permitirá aos usuários salvar ”vibes” ou configurações específicas de busca para reutilização futura, facilitando a curadoria recorrente.
- **Registro de Interações:** Expandir a coleta de dados para registrar sistematicamente todas as interações do usuário com o player e com as recomendações, criando um repositório rico de comportamento implícito e explícito.
- **Recomendação Baseada em Comportamento:** Utilizar o banco de interações coletadas para treinar um modelo de recomendação personalizado. Este modelo evoluirá além da similaridade de conteúdo, aprendendo ativamente as preferências individuais para gerar sugestões cada vez mais precisas.
- **Paginação de Recomendações:** Implementar a navegação paginada no motor de busca por vizinhos mais próximos, possibilitando que o usuário explore o vasto catálogo de forma sequencial, sem duplicação de resultados.