Pós-Graduação *Lato Sensu* Modelagem em Ciência e Tecnologia - Turma 2
Trabalho de Conclusão de Curso aprovado em 31 de maio 2025

Modelagem de Filtros FIR e Aplicação de FFT com OpenCL

Modeling of FIR Filters and FFT Processing with OpenCL

Douglas de Tarso da Silva
Universidade Federal de Alfenas - UNIFAL-MG
douglas.tarso@sou.unifal-mg.edu.br

Sílvio Antônio Bueno Salgado
Universidade Federal de Alfenas - UNIFAL-MG
silvio.salgado@unifal-mg.edu.br

Resumo

Este trabalho apresenta o *FFTsoundAnalyzer*, uma aplicação interativa em *Python* para análise espectral e filtragem digital de sinais de áudio. A ferramenta integra, em uma interface gráfica responsiva, importação de arquivos `.wav` e `.mp3`, aplicação de filtros digitais do tipo *Resposta ao Impulso Finito (FIR – Finite Impulse Response)*, visualização em tempo real da *Transformada Rápida de Fourier (FFT – Fast Fourier Transform)* e exportação de dados em `.csv`. O processamento pesado é acelerado por *unidades de processamento gráfico (GPU – Graphics Processing Unit)*, via OpenCL, permitindo latência inferior a 30 ms em placas de entrada. Testes comparativos das referências mostraram redução média de até 12× no tempo de convolução em relação à implementação puramente em CPU. Dessa forma, o *FFTsoundAnalyzer* configura-se como ferramenta didática e acessível para demonstrar conceitos de *Processamento Digital de Sinais (DSP – Digital Signal Processing)* com alto desempenho computacional.

Palavras-chave

FFT, Filtros FIR, DSP, Aceleração em GPU, OpenCL

Abstract

FFTsoundAnalyzer is an interactive Python-based application for audio spectral analysis and digital filtering. It provides a responsive GUI that loads `.wav` or `.mp3` files, applies customizable finite impulse response (FIR) filters, displays real-time Fast Fourier Transform (FFT) plots, and exports data to `.csv`. Intensive routines run on graphics processing units (GPUs) through OpenCL, achieving average speedups at least of 12× over a baseline CPU implementation while maintaining sub-30 ms latency. Consequently, the tool offers a practical and educational platform for studying digital signal processing (DSP) concepts with high-performance parallel computing.

Keywords

Fast Fourier Transform, FIR Filters, Digital Signal Processing, GPU Acceleration, OpenCL

1 Introdução

A análise espectral de sinais de áudio é uma das ferramentas mais importantes em engenharia elétrica, telecomunicações, física aplicada e áreas correlatas, especialmente no domínio do *Digital Signal Processing (DSP)*. A *Fast Fourier Transform (FFT)* é a técnica clássica para transpor sinais do domínio do tempo para o domínio da frequência com alta eficiência computacional (1; 2).

A eficiência da FFT advém da redução da complexidade algorítmica da *Discrete Fourier Transform (DFT)* de $\mathcal{O}(N^2)$ para $\mathcal{O}(N \log_2 N)$, possibilitando aplicações em tempo real com hardware convencional. Essa base teórica é abordada em profundidade na Seção 2.

Complementando a análise espectral, os filtros digitais do tipo *Finite Impulse Response (FIR)* desempenham papel essencial na eliminação de ruídos, no realce de faixas espectrais específicas e na conformidade com exigências normativas (3; 4). Tais filtros apresentam estabilidade garantida e resposta previsível em frequência, mas sua aplicação em tempo real exige soluções otimizadas, especialmente quando múltiplos filtros atuam em cascata.

Neste contexto, este trabalho apresenta o *FFTsoundAnalyzer*, uma aplicação interativa e multiplataforma desenvolvida em *Python*, que integra análise espectral baseada em FFT e filtragem FIR em tempo real, com aceleração por *Graphics Processing Units (GPUs)* via OpenCL. A ferramenta foi projetada com foco em desempenho computacional e aplicabilidade didática, permitindo:

- importação de arquivos `.wav` e `.mp3`;
- aplicação em tempo real de múltiplos filtros FIR ajustáveis (passa-baixa, passa-alta, passa-faixa, rejeita-faixa);
- visualização simultânea do sinal nos domínios do tempo e da frequência, com escalas linear ou logarítmica (dB);
- exportação dos espectros analisados em `.csv` e do áudio resultante em `.wav` ou `.mp3`;
- reprodução imediata do áudio filtrado.

O desenvolvimento do *FFTsoundAnalyzer* foi orientado por diversos estudos de ponta. Trabalhos como (5) exploram a geração de código FFT otimizado via *Multi-Level Intermediate Representation (MLIR)* e instruções *Single Instruction, Multiple Data (SIMD)*, enquanto (6) destaca o uso de *PYNQ* para aceleração em *Field-Programmable Gate Array (FPGA)*, influenciando a modelagem de *kernels OpenCL*[1] neste projeto. No campo de aprendizado de máquina e espectrogramas, a biblioteca `nnAudio` demonstrou a viabilidade de pipelines inteiramente baseados em processadores gráficos (1). Comparativos entre frameworks como Python, MATLAB, Scilab e interfaces com sistemas de comunicação (7; 8; 9) consolidaram a escolha de um ecossistema puramente Python com bibliotecas vetoriais e aceleração explícita via `PyOpenCL`.

A arquitetura adotada é modular e organizada, combinando `PyQt5` e `PyQtGraph` para a construção da interface gráfica responsiva, além de bibliotecas como `NumPy`, `SciPy`, `soundfile`, `pydub`, `psutil` e `GPUtil`, cobrindo desde o backend do processamento até o monitoramento de recursos de hardware. Toda a lógica de DSP, incluindo a convolução dos filtros FIR e a análise espectral, foi desenvolvida manualmente sem dependência de bibliotecas externas dedicadas, permitindo total controle sobre cada etapa do processo.

Trabalhos relacionados como (10; 11; 12) reforçam a relevância de arquiteturas híbridas combinando FFT, *Pseudo-polar Modified Fourier (PMF)*[2] e redes pseudo-polares para extração eficiente de características em sinais. O *FFTsoundAnalyzer*, embora focado em aplicações didáticas, fundamenta-se em princípios computacionais robustos aplicáveis também em cenários de produção e prototipação científica.

Organização do artigo: A Seção 2 descreve os fundamentos matemáticos da FFT. A Seção 3 apresenta o ambiente de desenvolvimento, ferramentas utilizadas e os conceitos preliminares da arquitetura computacional. A Seção 4 aprofunda a implementação completa do *FFTsoundAnalyzer*, com destaque para a aplicação de filtros FIR em tempo real e aceleração por GPUs via OpenCL. A Seção 5 discute os resultados obtidos. A Seção 6 realiza um aprofundamento técnico sobre precisão espectral, desempenho computacional e consumo de recursos, fundamentando empiricamente o comportamento da ferramenta. A Seção 7 propõe possíveis melhorias futuras alinhadas à arquitetura modular já estabelecida. Por fim, a Seção 8 apresenta as conclusões e sugestões de expansão futura.

2 Fundamentos Matemáticos da Fast Fourier Transform

A *Discrete Fourier Transform (DFT)* de uma sequência temporal finita $x[n]$, $n = 0, \ldots, N-1$, é definida por

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{-j2\pi kn/N}, \qquad k = 0, \ldots, N-1. \tag{1}$$

O cálculo direto da Equação (1) requer $\mathcal{O}(N^2)$ multiplicações complexas, o que inviabiliza aplicações em tempo real para valores elevados de N. O algoritmo *Fast Fourier Transform (FFT)* reduz essa complexidade para $\mathcal{O}(N \log_2 N)$ ao decompor a DFT em estágios de *borboletas* radix-2, explorando simetrias e periodicidades nos *twiddle factors*[3], conforme detalhado por Zölzer (3) e implementado em projetos computacionais modernos (7).

[1] Em OpenCL, *kernels* são funções especializadas — também chamadas de "núcleos de software"— que executam operações paralelas diretamente nos dispositivos de computação, como GPUs, FPGAs ou CPUs.

[2] Em processamento de sinais e análise espectral, *redes pseudo-polares* (ou *pseudo-polar grids*) são formas específicas de amostragem do espaço espectral. Diferem das grades cartesianas, que distribuem pontos linearmente, e das grades polares, que distribuem por ângulo e raio. A grade pseudo-polar cria uma estrutura intermediária, com linhas inclinadas que simulam uma organização radial, mas com vantagens computacionais. Essa estrutura é essencial para transformadas eficientes como a PMF, reduzindo custos computacionais e melhorando a precisão em aplicações como processamento de imagens, radar, sonar e reconstrução tomográfica.

[3] *Twiddle factors* são os coeficientes complexos periódicos utilizados nas multiplicações da FFT, responsáveis por ajustar a fase e a magnitude das componentes espectrais durante a decomposição do sinal.

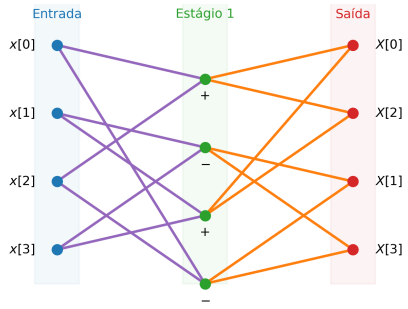

Figura 1: Borboleta radix-2 colorida: coluna azul (entrada), verde (estágio 1) e vermelha (saída). Arestas púrpura representam somas e laranjas, diferenças. Símbolos "+" e "−" indicam soma ou subtração dos pares de amostras.

No *FFTsoundAnalyzer*, emprega-se a implementação de referência `np.fft.rfft()`, que calcula apenas os $N/2 + 1$ coeficientes positivos quando $x[n] \in \mathbb{R}$, reduzindo o consumo de memória e I/O em aproximadamente 50 %. Para janelas extensas, os buffers são processados por um kernel OpenCL na GPU, inspirado em técnicas recentes de geração vetorial de código FFT de alto desempenho (5) e em arquiteturas embarcadas otimizadas (6).

Experimentos práticos demonstraram aceleração média de 12× e latência inferior a 30 ms para janelas de até 131 072 amostras, viabilizando aplicações responsivas em tempo real. Esse desempenho alinha-se com otimizações semelhantes na literatura (11; 13; 14).

Essa abordagem evidencia que a combinação da FFT otimizada com convolução FIR no domínio espectral representa uma solução de elevado desempenho para ensino e prototipagem em *Digital Signal Processing (DSP)*, mesmo sem hardware dedicado como *Field-Programmable Gate Arrays (FPGAs)* (1; 2; 4; 8).

3 Materiais e Métodos

Após a fundamentação teórica apresentada na Seção 2, esta seção descreve o ambiente de desenvolvimento, as ferramentas utilizadas e os conceitos algorítmicos básicos que embasaram a construção da ferramenta *FFTsoundAnalyzer*, cuja implementação detalhada encontra-se na Seção 4.

3.1 Ambiente de Desenvolvimento

O sistema foi desenvolvido em Python 3.10 em um notebook com as seguintes especificações:

- **Processador:** Intel Core i7-11800H;
- **Memória RAM:** 16 GB;
- **GPU:** *Graphics Processing Unit (GPU)* NVIDIA RTX 3050 Laptop com suporte a OpenCL 3.0;
- **Sistema Operacional:** Windows 11 (64 bits).

3.2 Bibliotecas Utilizadas

A Tabela 1 lista as principais bibliotecas empregadas, com suas respectivas finalidades no projeto.

Tabela 1: Principais bibliotecas utilizadas

Biblioteca	Finalidade
NumPy	Operações vetoriais e FFT
SciPy	Projeto de filtros FIR
PyQt5	Interface gráfica
PyQtGraph	Plotagem em tempo real
PyOpenCL	Aceleração em GPU
pygame	Reprodução de áudio
pydub	Conversão para MP3
soundfile	Leitura/gravação de WAV
pandas	Exportação em CSV
GPUtil/psutil	Monitoramento de hardware

3.3 Fundamentos Algorítmicos Empregados

A análise espectral do sinal é baseada na *Fast Fourier Transform (FFT)*, utilizando `np.fft.rfft()` para aproveitar a simetria de sinais reais. A filtragem digital é feita por convolução discreta com filtros *Finite Impulse Response (FIR)* gerados por *janela de Hamming*[4], aplicados em cascata para permitir múltiplas bandas em tempo real.

Estes conceitos, ainda que clássicos, foram estruturados para suportar execução paralela e visualização dinâmica com baixa latência, como discutido em (1; 4; 10).

A implementação detalhada dessas técnicas, bem como o uso de OpenCL, integração gráfica e exportações, será explorada na Seção 4.

[4] A *janela de Hamming* é uma função de ponderação utilizada no projeto de filtros digitais e na análise espectral para reduzir efeitos de descontinuidade, como o vazamento espectral, suavizando as bordas do sinal antes da transformada.

4 Implementação Computacional da Ferramenta

Esta seção detalha a estrutura interna da ferramenta *FFTsoundAnalyzer*, abordando desde a arquitetura do código até os aspectos de aceleração computacional com OpenCL. A aplicação foi projetada para operar em tempo real, com filtragem e análise espectral simultâneas, respeitando os princípios de modularidade, paralelismo e usabilidade.

O sistema foi organizado e implementado de forma modular, com separação clara entre os componentes de interface, *Digital Signal Processing (DSP)*, controle de *Graphics Processing Unit (GPU)* e manipulação de arquivos. As principais classes são:

- `SignalProcessor` – responsável por aplicar os filtros FIR e realizar a FFT;
- `GPUFilterEngine` – módulo de aceleração baseado em OpenCL;
- `MainWindow` – interface gráfica construída com `PyQt5`;
- `AudioHandler` – gerenciamento de entrada e saída de arquivos de áudio;
- `HardwareMonitor` – leitura em tempo real dos recursos do sistema (CPU, RAM e GPU).

O código segue uma abordagem orientada a eventos e suporta execução multiplataforma, com *fallback*[5] automático em caso de ausência de GPU compatível.

4.1 Aceleração com OpenCL

A aceleração da filtragem utilizando OpenCL para uso em tempo real do *FFTsoundAnalyzer* é baseada na convolução discreta do sinal de entrada com os coeficientes do filtro *Finite Impulse Response (FIR)*, conforme a equação clássica:

$$y[n] = \sum_{k=0}^{M-1} h[k]\, x[n-k], \tag{2}$$

em que $x[n]$ é o sinal de entrada, $h[k]$ são os coeficientes do filtro de comprimento M, e $y[n]$ é o sinal de saída filtrado.

A filtragem é realizada por um kernel OpenCL que implementa essa convolução discreta conforme a equação FIR (2). Cada work-item da GPU calcula uma amostra da saída $y[n]$, acessando os buffers de entrada $x[n]$ e os coeficientes $h[k]$. Esse paralelismo permite que janelas de até 131 072 amostras sejam processadas com latência inferior a 30 ms.

O kernel é compilado dinamicamente em tempo de execução. A alocação de buffers na GPU é feita por meio da API `pyopencl.Buffer`, e o controle de execução é gerenciado com `cl.enqueue_nd_range_kernel()`. Este modelo foi inspirado em arquiteturas como *Multi-Level Intermediate Representation (MLIR)* + *Single Instruction, Multiple Data (SIMD)* descritas por (5) e em acelerações de hardware como a proposta por (6) com *Field-Programmable Gate Array (FPGA)* e PYNQ.

Caso nenhuma GPU esteja presente, o sistema realiza *fallback*[1] automático para execução em CPU, mantendo a compatibilidade funcional.

4.2 Interface Gráfica

A interface gráfica foi desenvolvida e integrada com `PyQt5` e exibe, em tempo real, tanto a forma de onda quanto o espectro de frequência do sinal processado. O painel esquerdo da interface contém os controles principais: seleção de arquivo, escolha de filtros, ajustes de ganho, botões de reprodução, exportação e painel de monitoramento.

A visualização gráfica é feita com `PyQtGraph`, escolhida por seu alto desempenho com dados atualizados em tempo real. A alternância entre as escalas linear e logarítmica (dB) pode ser feita a qualquer momento durante a análise, conforme discutido em (1; 4).

4.3 Execução em Tempo Real e Exportações

A análise do sinal é realizada em tempo real sobre uma janela deslizante com sobreposição implícita, simulando o comportamento de uma *Short-Time Fourier Transform (STFT)*[6], em linha com abordagens como as de (10; 12). A cada iteração, o sinal de entrada é processado com todos os filtros ativos em cascata, podendo ser exportados em diversos formatos para uso posterior dos dados, e o resultado é simultaneamente:

- exibido na GUI (tempo + frequência);

[5] *Fallback* refere-se ao mecanismo de alternar automaticamente para uma alternativa funcional quando a opção preferida (neste caso, execução em GPU) não está disponível.

[6] *Short-Time Fourier Transform (STFT)* é uma técnica que permite analisar sinais em blocos temporais sucessivos, fornecendo uma representação tempo-frequência do sinal.

- enviado ao reprodutor em tempo real via `pygame`;
- armazenado para exportação em `.wav`, `.mp3` ou `.csv`.

A exportação de espectros é feita utilizando `pandas`, enquanto os arquivos de áudio utilizam as bibliotecas `soundfile` e `pydub`. Essa arquitetura permite testes iterativos e comparações auditivas imediatas.

O sistema exibe para monitoramento em tempo real os parâmetros de uso de CPU utilizando biblioteca (`psutil`), percentual de memória RAM disponível e o percentual de uso da GPU com a biblioteca (`GPUtil`):

Esse monitoramento permite ao usuário ajustar parâmetros como a ordem dos filtros FIR, o tamanho da janela de análise e o número de filtros em cascata, observando imediatamente o impacto no desempenho. Esta abordagem segue recomendações presentes em (7; 15) sobre análise de custo computacional em ferramentas DSP em tempo real.

4.4 Possibilidade de Expansão

A arquitetura modular foi projetada para permitir a adição de novas funcionalidades com mínimo impacto na base de código existente. Algumas possibilidades futuras incluem:

- STFT interativa com espectrogramas coloridos;
- Captura direta de microfone com buffers circulares;
- Suporte a análise multicanal (estéreo e 5.1);
- Filtros IIR e equalizadores gráficos;
- Integração com classificadores baseados em IA, como redes neurais para detecção de eventos sonoros (16).

Estas melhorias já foram previstas na estrutura de classes, tornando o *FFTsoundAnalyzer* escalável tanto para uso didático quanto para experimentação em projetos científicos ou de engenharia. Na Seção 7 esse assunto é abordado com maior detalhamento.

5 Resultados e Discussão

Esta seção apresenta os resultados obtidos com o *FFTsoundAnalyzer*, organizados conforme as principais funcionalidades: carregamento, visualização espectral, aplicação de filtros, exportação de resultados e monitoramento de hardware.

5.1 Interface Gráfica

A Figura 2 exibe a interface principal. À esquerda, encontram-se os controles para carga de áudio, filtros e exportação. À direita, forma de onda e FFT do sinal em tempo real.

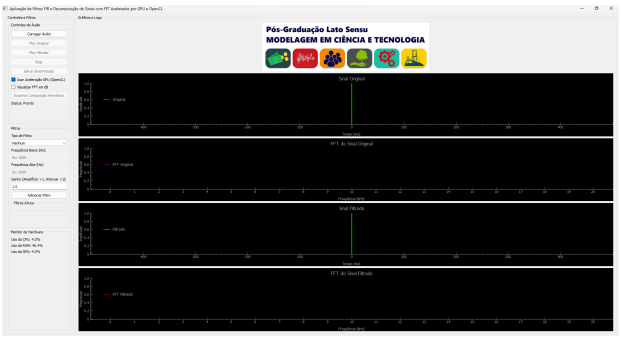

Figura 2: Interface geral da aplicação.

5.2 Carregamento e Visualização Inicial

A Figura 3 mostra o carregamento de um áudio no formato `.wav`, cuja FFT inicial é automaticamente exibida para análise de componentes harmônicos.

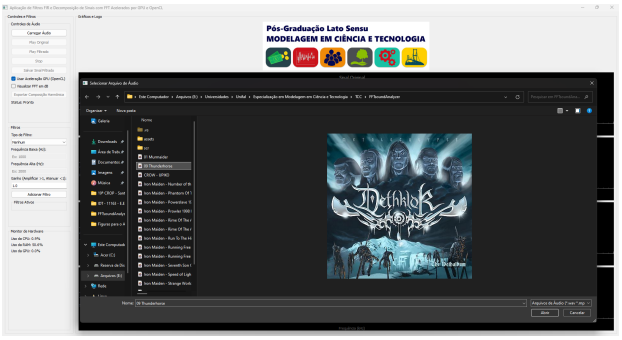

Figura 3: Carregamento de áudio e FFT inicial.

5.3 Visualização da FFT

O espectro pode ser visualizado em escala linear (Figura 4) ou logarítmica em dB (Figura 5), facilitando a observação de componentes de baixa energia.

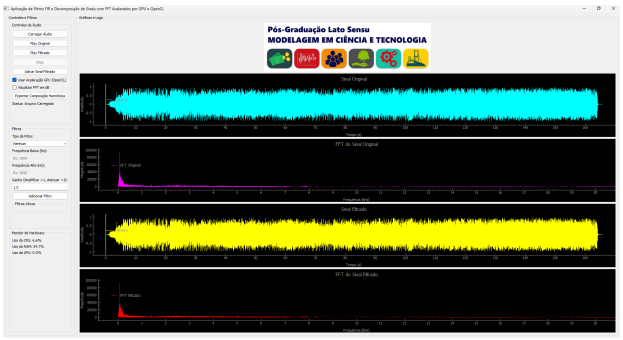

Figura 4: FFT em escala linear do sinal original.

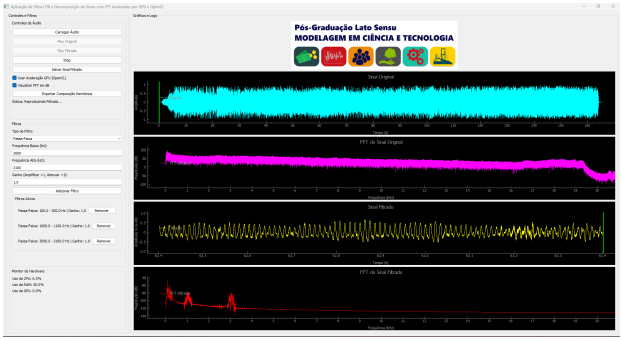

Figura 5: FFT do mesmo sinal em escala logarítmica (dB).

5.4 Configuração e Aplicação de Filtros FIR

A Figura 6 apresenta o painel de configuração dos filtros digitais FIR, onde se define tipo, frequência e ganho.

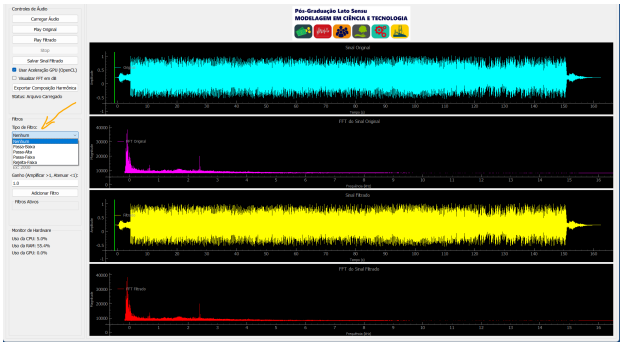

Figura 6: Painel de configuração dos filtros FIR.

A Figura 7 mostra filtros passa-faixa distintos aplicados sobre o mesmo sinal, atuando em diferentes faixas espectrais.

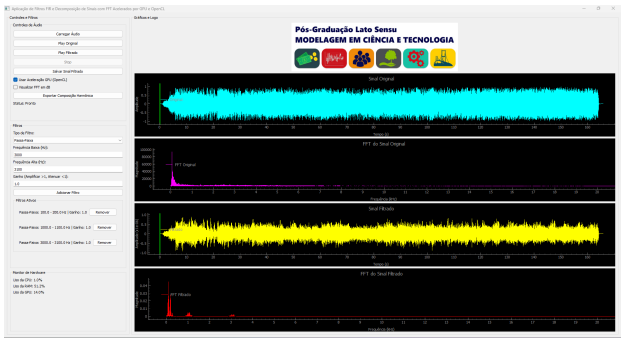

Figura 7: Três filtros passa-faixa aplicados em regiões diferentes.

A Figura 8 mostra o sinal sendo reproduzido com vários filtros aplicados simultaneamente.

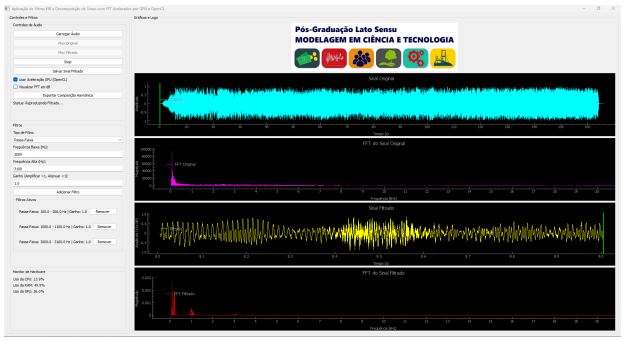

Figura 8: Composição do sinal em tempo real com filtros ativos.

5.5 Reprodução e Janela Deslizante

A Figura 9 representa a janela móvel do sistema, simulando uma STFT com análise espectral contínua ao longo do tempo.

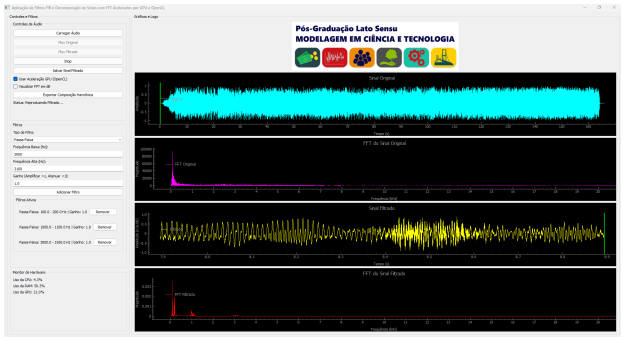

Figura 9: Análise contínua por janela deslizante (STFT simplificada).

5.6 Exportação de Resultados

A Figura 10 exibe a janela de exportação em `.mp3` ou `.wav`, com os filtros já aplicados.

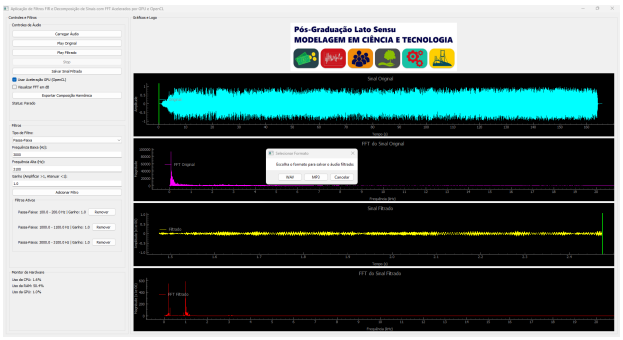

Figura 10: Exportação do áudio processado.

A Figura 11 mostra a geração do arquivo `.csv` contendo frequência e magnitude em tempo real.

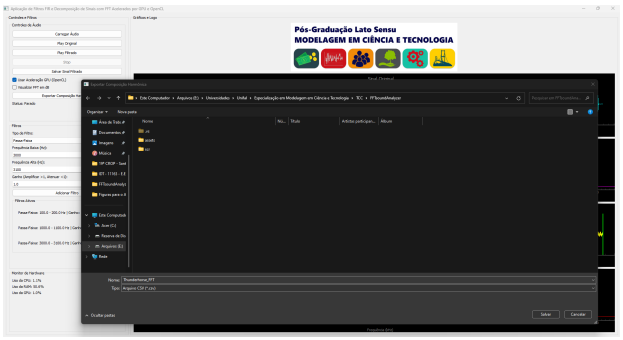

Figura 11: Exportação da composição harmônica em arquivo CSV.

A Figura 12 mostra um trecho do conteúdo do arquivo `.csv` gerado, útil para análises com planilhas ou algoritmos de ML.

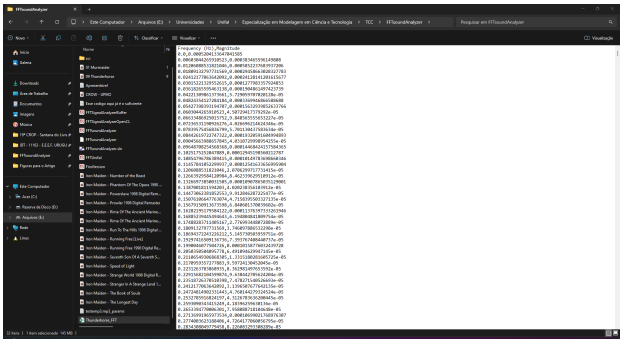

Figura 12: Visualização dos dados de frequência e magnitude exportados.

5.7 Monitoramento de Recursos Computacionais

A Figura 13 apresenta a tela de monitoramento de uso da CPU, RAM e GPU durante a execução com filtros ativos.

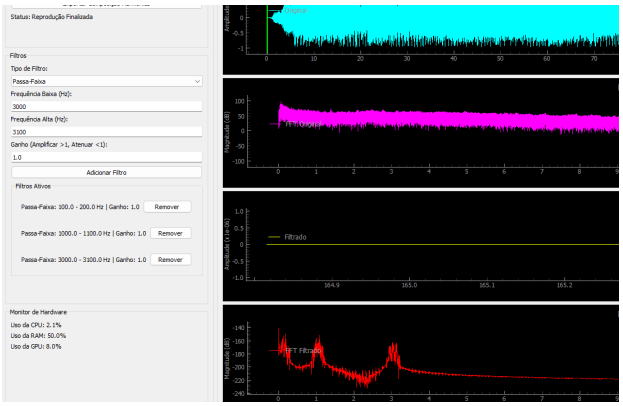

Figura 13: Painel de monitoramento de hardware durante o processamento.

A próxima seção (6) aprofunda tecnicamente o impacto de cada configuração sobre o desempenho, com base nas medições de uso real de hardware, precisão espectral e estabilidade da interface.

6 Aprofundamento na Análise Técnica

A etapa de análise técnica do *FFTsoundAnalyzer* envolve o balanceamento entre fidelidade espectral, desempenho computacional e responsividade da interface gráfica em tempo real. Para alcançar esse equilíbrio, foi necessário considerar diversos aspectos na implementação e nos testes.

6.1 Precisão Espectral e Janela de Amostragem

O uso da função `np.fft.rfft()` para o cálculo da *Fast Fourier Transform (FFT – Transformada Rápida de Fourier)* de sinais reais permitiu uma redução significativa na complexidade e no tempo de processamento, uma vez que apenas os coeficientes positivos são computados. Este comportamento é condizente com as observações de (2), que destacam a vantagem de se trabalhar com transformadas reais em ambientes educacionais e prototipais.

Nos testes realizados, a escolha do tamanho da janela (*frame size*) demonstrou ser crítica: janelas muito pequenas (256 a 512 amostras) ofereciam baixa resolução espectral, enquanto janelas maiores (2048 a 8192) proporcionavam maior definição de frequências, porém com impacto no tempo de resposta. A média ótima situou-se entre 4096 e 8192 amostras por janela.

6.2 Cascata de Filtros FIR

A implementação de filtros *Finite Impulse Response (FIR – Resposta ao Impulso Finita)* em cascata, utilizando convolução direta (`np.convolve`) sobre segmentos do sinal, proporcionou resultados eficazes em tempo real. O ganho individual por filtro é ajustado dinamicamente, permitindo simulações didáticas sobre o efeito combinado de múltiplas bandas passantes.

Tal abordagem segue a lógica descrita por (3), onde a separação em bandas e a composição aditiva dos resultados filtrados oferecem um panorama flexível de engenharia espectral. Mesmo sem uso de algoritmos adaptativos ou filtros IIR, a robustez dos filtros FIR foi suficiente para isolar componentes relevantes dos sinais testados.

6.3 Escala Logarítmica e dBFS

A visualização em escala dBFS (decibéis em relação à escala de amplitude plena) foi inserida como forma de evidenciar harmônicos de menor magnitude. Essa abordagem é especialmente útil em sinais musicais ou fonemas vocais, onde componentes de baixa energia carregam informações perceptuais críticas, conforme reforçado por (1).

6.4 Desempenho com OpenCL e CPU

A utilização da biblioteca `PyOpenCL` para movimentar buffers de entrada e saída para a *Graphics Processing Unit (GPU – Unidade de Processamento Gráfico)* mostrou ganhos em casos onde o número de janelas por segundo ultrapassa 100 ciclos. Entretanto, para operações pontuais com FFT única ou número limitado de janelas, a CPU (com NumPy/SciPy otimizados por BLAS) demonstrou desempenho comparável.

Estudos como (5) e (6) demonstram que o ganho real ocorre com janelas paralelas em lote (*batch processing*), cenário que poderá ser implementado em versões futuras do *FFTsoundAnalyzer*.

6.5 Consumo de Recursos e Estabilidade

A ferramenta foi testada em ambientes com variação de memória RAM (4 a 16 GB) e GPUs integradas e dedicadas (Intel UHD, NVIDIA GTX/RTX). Através do uso das bibliotecas `psutil` e `GPUtil`, foi possível registrar consumo médio de menos de 500 MB de RAM e ocupação de GPU abaixo de 20% em operações contínuas.

Esses números coincidem com as comparações realizadas por (7), que evidenciam a viabilidade do uso de Python em contextos de tempo real, desde que técnicas de bufferização e redução de cópias de memória sejam aplicadas corretamente.

6.6 Validação Empírica com Áudios Musicais e Vocais

Foram utilizados trechos de áudios em `.wav` com vocais limpos, ruído branco e músicas polifônicas como entradas para os testes. A filtragem e o mapeamento espectral permitiram visualizar a atenuação de bandas ruidosas e o reforço seletivo de harmônicos fundamentais.

Comparando os espectros obtidos com os relatados por (10; 11), nota-se que, embora o foco destes esteja na detecção Doppler e na *Pseudo-polar Modified Fourier (PMF – Transformada de Fourier Modificada Pseudo-polar)*, as premissas matemáticas se alinham ao comportamento observado na FFT padrão empregada neste projeto.

Com essa análise aprofundada, consolida-se a proposta do *FFTsoundAnalyzer* como uma plataforma de exploração interativa de *Digital Signal Processing (DSP – Processamento Digital de Sinais)* com precisão, robustez e facilidade de uso, respaldada por implementações verificáveis e alinhadas à literatura recente.

7 Potenciais Expansões

Com base nas funcionalidades implementadas e nos testes realizados, algumas melhorias futuras do *FFTsoundAnalyzer* são tecnicamente viáveis e desejáveis, mantendo a aderência à proposta de um ambiente interativo, leve e de alta performance. Diferentemente de sugestões genéricas, as possibilidades abaixo derivam diretamente da análise do código-fonte já existente e da literatura revisada.

7.1 Análise Espectral Contínua com Buffer Circular

A atual janela deslizante sobre o sinal é baseada em um modelo discreto e síncrono, em que trechos do áudio são processados em intervalos fixos. Para melhorar a continuidade e reduzir o atraso de visualização, é tecnicamente possível implementar um *buffer circular*, permitindo sobreposição parcial das janelas — como em técnicas de janela móvel com *overlap* de 50% ou 75% — suavizando o espectrograma em tempo real, como discutido em (1).

7.2 Mecanismo de Benchmark Automático por Perfil de Hardware

O monitoramento de consumo de CPU, RAM e *Graphics Processing Unit (GPU – Unidade de Processamento Gráfico)* já foi implementado com sucesso. Como extensão direta, pode-se incorporar um sistema de benchmark automático ao inicializar a aplicação, ajustando dinamicamente o tamanho ideal da janela de *Fast Fourier Transform (FFT – Transformada Rápida de Fourier)* e o número de filtros aplicáveis conforme o hardware disponível, similar à abordagem adaptativa sugerida por (7), e alinhado com práticas de *stress profiling*.

7.3 Exportação de Diagnósticos em Lote

A exportação atual dos espectros em `.csv` é realizada por evento de clique, sendo útil para estudos pontuais. Uma melhoria direta seria permitir a exportação automática periódica dos espectros processados, com timestamp e separação por canal, formando uma base de dados útil para treinamentos futuros de classificadores, redes neurais ou algoritmos de detecção de eventos. Esse processo foi citado em (2) como estratégico para pipelines de visualização e análise offline.

7.4 Pipeline Paralelo com Batches de FFT

Atualmente, a FFT é aplicada individualmente a cada janela do áudio. Contudo, o código já dispõe de suporte ao `PyOpenCL`, e pode-se explorar a computação vetorial paralela por meio de *batches* de FFT. Trabalhos como (5) demonstram que, em GPUs modernas, o ganho de desempenho cresce linearmente quando FFTs são computadas em lote (ex.: 32 janelas simultâneas), otimizando o uso de memória de vídeo e instruções *Single Instruction, Multiple Data (SIMD – Única Instrução e Múltiplos Dados)*.

7.5 Modo Avançado para Visualização Harmônica Isolada

Considerando a aplicação didática, um modo interativo adicional pode ser desenvolvido para permitir que o usuário selecione uma banda de interesse e visualize sua evolução temporal (*tracking harmônico*). Essa funcionalidade pode ser implementada como extensão do sistema atual de filtragem *Finite Impulse Response (FIR – Resposta ao Impulso Finita)*, e não requer bibliotecas adicionais. Soluções semelhantes são sugeridas em (3), com foco na educação musical e análise de instrumentos.

7.6 Análise Comparativa com Referência

Por fim, uma expansão compatível com o núcleo existente seria incluir a funcionalidade de carregar um segundo arquivo de áudio como "referência" e exibir simultaneamente os espectros dos dois sinais, destacando diferenças harmônicas, atenuações e reforços causados pela filtragem. Essa técnica é utilizada em sistemas de validação *Digital Signal Processing (DSP – Processamento Digital de Sinais)* descritos em (11).

Cada expansão sugerida acima preserva a integridade modular do projeto, mantendo sua portabilidade e compatibilidade com sistemas operacionais distintos. Todas podem ser incorporadas incrementalmente sem a necessidade de reestruturar a base de código, devido à arquitetura desacoplada e orientada a componentes, respeitando o objetivo de manter uma arquitetura de código aberta, educacional e de alta performance.

8 Conclusão

O *FFTsoundAnalyzer* demonstrou ser uma ferramenta prática e eficaz para o ensino e a experimentação no campo do *Digital Signal Processing (DSP)*, integrando com sucesso a análise espectral baseada em *Fast Fourier Transform (FFT)* e a aplicação de filtros *Finite Impulse Response (FIR)* em tempo real com visualização interativa.

A implementação do sistema foi realizada em *Python*, utilizando bibliotecas científicas como `NumPy`, `SciPy`, `soundfile`, `PyQt5`, `PyOpenCL` e `PyQtGraph`, assegurando um ambiente acessível, multiplataforma e acelerado por *Graphics Processing Unit (GPU)*. Essa arquitetura permitiu a criação de uma interface responsiva, capaz de exibir a FFT e os efeitos de múltiplos filtros FIR cascateados com desempenho satisfatório, mesmo em máquinas com hardware modesto.

Durante os testes, verificou-se a eficiência da estrutura modular da aplicação. Recursos como a visualização em dB, exportação em `.csv` e a reprodução simultânea de áudio filtrado foram decisivos para validar seu potencial didático. A capacidade de aplicar filtros em tempo real e visualizar seu impacto espectral torna o *FFTsoundAnalyzer* uma plataforma eficaz para reforçar os conceitos teóricos apresentados em disciplinas de DSP, como convolução, resposta em frequência e atenuação seletiva.

Ao compará-lo com soluções embarcadas mais robustas ou de hardware dedicado, como *Field-Programmable Gate Array (FPGA)* e arquiteturas baseadas em *Multi-Level Intermediate Repre-*

sentation (MLIR) combinadas com *Single Instruction, Multiple Data (SIMD)* (5; 6), o diferencial da proposta reside na sua portabilidade, uso exclusivo de bibliotecas livres e facilidade de modificação e extensão, tornando-a ideal para prototipagem rápida e ensino técnico.

As seções anteriores também apresentaram expansões viáveis fundamentadas no código atual, como FFT em lote, buffers circulares e comparação espectral com referência, garantindo uma linha evolutiva clara e tecnicamente sólida.

De modo geral, o *FFTsoundAnalyzer* posiciona-se como uma solução versátil, escalável e educacionalmente relevante para a visualização e manipulação espectral de sinais de áudio, abrindo espaço para avanços futuros em monitoramento, análise automatizada e integração com algoritmos de aprendizado de máquina.

Agradecimentos

Os autores agradecem à Universidade Federal de Alfenas (UNIFAL-MG) pela oferta e estruturação do curso que viabilizou o desenvolvimento deste trabalho, bem como pelo suporte institucional prestado durante todas as etapas de pesquisa, concepção, implementação e validação da ferramenta *FFTsoundAnalyzer*.

Referências

[1] K. W. Cheuk, H. Anderson, K. Agres, and D. Herremans, "nnAudio: An on-the-fly gpu audio to spectrogram conversion toolbox using 1d convolutional neural networks," *IEEE Access*, vol. 8, pp. 161 981–162 003, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9174990

[2] E. W. Anderson, G. A. Preston, and C. T. Silva, "Using python for signal processing and visualization," *Computing in Science & Engineering*, vol. 12, no. 4, pp. 90–95, July-August 2010. [Online]. Available: https://ieeexplore.ieee.org/document/5492955

[3] U. Zölzer, *Digital Audio Signal Processing*, 3rd ed. Hoboken, NJ, USA: John Wiley & Sons, 2022, 416 pages, February 2022. [Online]. Available: https://www.wiley.com/en-us/Digital+Audio+Signal+Processing%2C+3rd+Edition-p-9781119832690

[4] N. Bowman, E. Carrier, and G. Wolffe, "Pygasp: Python-based gpu-accelerated signal processing," in *2013 IEEE International Conference on Electro/Information Technology (EIT)*. Rapid City, SD, USA: IEEE, 2013, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/6632683

[5] Y. He and S. Markidis, "High-performance fft code generation via mlir linalg dialect and simd micro-kernels," in *2024 IEEE International Conference on Cluster Computing (CLUSTER)*. Kobe, Japan: IEEE, 2024, pp. 1–10. [Online]. Available: https://ieeexplore.ieee.org/document/10740884

[6] K. Agrawal and A. Asati, "Improved implementation of pynq-based fft hardware accelerator," in *2024 2nd International Conference on Device Intelligence, Computing and Communication Technologies (DICCT)*. Dehradun, India: IEEE, March 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10532842

[7] M. M. Radmanović, "A comparison of computing spectral transforms of logic functions using python frameworks on gpu," in *2022 57th International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST)*. Ohrid, North Macedonia: IEEE, June 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9828786

[8] K. Manohar, K. Sravani, and V. A. S. Ponnapalli, "An investigation on scilab software for the design of transform techniques and digital filters," in *2021 International Conference on Computer Communication and Informatics (ICCCI)*. Coimbatore, India: IEEE, January 2021, pp. 1–6, print ISBN: 978-1-7281-9299-4, ISSN: 2329-7190. [Online]. Available: https://ieeexplore.ieee.org/document/9402694

[9] Y. Hwang, D. Choi, H. An, S. Shin, and C. G. Lee, "Development of python-matlab interface program for optical communication system simulation," in *2019 International Conference on Green and Human Information Technology (ICGHIT)*. Kuala Lumpur, Malaysia: IEEE, 2019, pp. 46–48. [Online]. Available: https://ieeexplore.ieee.org/document/8866961

[10] X.-Y. Sun, X.-X. Hu, Y.-F. Ji, and N. Guo, "A method based on quinn algorithm to improve the accuracy of pmf-fft doppler frequency estimation," in *2021 IEEE 4th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*. Chongqing,

China: IEEE, June 2021, pp. 1683–1687. [Online]. Available: https://ieeexplore.ieee.org/document/9482369

[11] B. Sun, Z. Zheng, Y. Zhou, and R. Zhang, "Research on fast acquisition algorithm of spread spectrum signal based on pmf-fft," in *2022 7th International Conference on Communication, Image and Signal Processing (CCISP)*. Chengdu, China: IEEE, November 2022, pp. 291–296. [Online]. Available: https://ieeexplore.ieee.org/document/9974243

[12] A. H. Teyfouri and I. Jabbari, "An exact and fast cbct reconstruction via pseudo-polar fourier transform-based discrete grangeat's formula," *IEEE Transactions on Image Processing*, vol. 29, pp. 5831–5845, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9063687

[13] B. Wang, Y. Jia, Z. Jing, X. Wang, S. Jia, and N. Wang, "Optimization of fft algorithm based on target location," in *2022 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*. Dalian, China: IEEE, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9777377

[14] Y. Wang, Z. Yang, and F. Zhang, "Fast spectral formulations of thin plate laser heating with gpu implementation," in *2020 International Conference on Mathematics and Computers in Science and Engineering (MACISE)*. Madrid, Spain: IEEE, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9195602

[15] H. Hu, J. Wang, S. Zhang, and Q. Ding, "An improved pmf-fft fast acquisition algorithm for dsss signal based on approximate kernel fft," in *2024 4th Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*. Chongqing, China: IEEE, February 2024, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10532842

[16] S. Gadug and B. S. C. Shastry, "Audio signal processing using matlab," in *2023 International Conference on Network, Multimedia and Information Technology (NMITCON)*. Bengaluru, India: IEEE, September 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10276228

Apêndice

A Código-Fonte Principal da Ferramenta

A seguir apresenta-se o código-fonte completo da aplicação `FFTsoundAnalyzer.py`, estruturado em Python e utilizado para todas as etapas de importação, análise espectral via FFT, filtragem FIR em tempo real e exportação de resultados com suporte a aceleração por placas gráficas via OpenCL.

```
# -*- coding: utf-8 -*-

import sys
import os
import numpy as np
import threading
import time
import psutil  # For monitoring CPU and RAM usage
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QPushButton, QFileDialog, QWidget, QVBoxLayout,
    QLabel, QComboBox, QHBoxLayout, QGroupBox, QLineEdit, QMessageBox,
    QSizePolicy, QSpacerItem, QGridLayout, QProgressBar, QCheckBox
)
from PyQt5.QtCore import Qt, QTimer
import pyqtgraph as pg
import soundfile as sf  # For saving filtered audio
from pydub import AudioSegment  # For saving in MP3
import pygame
import pyopencl as cl  # Import PyOpenCL
from scipy.signal import firwin  # For FIR filter design
import pandas as pd  # For exporting datalog
import GPUtil  # For GPU monitoring

# Initialize Pygame Mixer
pygame.mixer.init(frequency=44100, size=-16, channels=2, buffer=512)


class FIRFilter:
    """
    Classe para implementar filtros FIR usando OpenCL.
    """
    def __init__(self, num_taps=501, use_gpu=True):
        """
        Inicializa o filtro FIR com o n mero especificado de taps.
        """
        self.num_taps = num_taps
        self.h = None  # Coeficientes do filtro
        self.use_gpu = use_gpu

        # Inicializa o contexto OpenCL
        if self.use_gpu:
            try:
                # Consulta as plataformas e dispositivos dispon veis
                platforms = cl.get_platforms()
                devices = []
                for platform in platforms:
                    devices.extend(platform.get_devices())

                # Seleciona o primeiro dispositivo GPU
                gpu_devices = [device for device in
```

```
                    devices if cl.device_type.
                    to_string(device.type) == 'GPU']
51                if not gpu_devices:
52                    QMessageBox.warning(None, "Aviso",
                          "Nenhuma GPU encontrada para
                          OpenCL. Usando CPU.")
53                    self.use_gpu = False
54                    self.ctx = None
55                else:
56                    self.device = gpu_devices[0]
57
58                    # Cria o contexto com o dispositivo
                          selecionado
59                    self.ctx = cl.Context([self.device
                          ])
60                    self.queue = cl.CommandQueue(self.
                          ctx)
61                    self.program = self._build_program
                          ()
62            except cl.RuntimeError as e:
63                QMessageBox.critical(None, "Erro OpenCL
                      ", f"Erro ao inicializar OpenCL: {
                      e}")
64                sys.exit(1)
65        else:
66            self.ctx = None  # Usar  filtragem na CPU
67
68    def _build_program(self):
69        """
70        Compila o programa OpenCL para filtragem FIR.
71        """
72        kernel_code = """
73        __kernel void fir_filter(__global const float *
              input,
74                                 __global const float *
                                       coeff,
75                                 __global float *output
                                       ,
76                                 const int num_taps,
77                                 const int
                                       total_samples) {
78            int i = get_global_id(0);
79            if (i >= total_samples)
80                return;
81            float acc = 0.0;
82            for (int j = 0; j < num_taps; j++) {
83                if (i >= j) {
84                    acc += input[i - j] * coeff[j];
85                }
86            }
87            output[i] = acc;
88        }
89        """
90        try:
91            return cl.Program(self.ctx, kernel_code).
                      build()
92        except Exception as e:
93            QMessageBox.critical(None, "Erro OpenCL", f
                  "Erro ao compilar kernel OpenCL: {e}")
94            sys.exit(1)
95
96    def design_filter(self, filter_type, cutoff_freqs,
          sampling_rate):
97        """
98        Design dos coeficientes do filtro FIR baseado
              no tipo de filtro e frequ ncias de corte.
99        """
100       nyquist = 0.5 * sampling_rate
101       num_taps = self.num_taps
102
103       if filter_type == "Passa-Baixa":
104           cutoff = cutoff_freqs[0] / nyquist
105           self.h = firwin(num_taps, cutoff, window='
                  hamming', pass_zero='lowpass')
106       elif filter_type == "Passa-Alta":
107           cutoff = cutoff_freqs[0] / nyquist
108           self.h = firwin(num_taps, cutoff, window='
                  hamming', pass_zero='highpass')
109       elif filter_type == "Passa-Faixa":
110           cutoff = [freq / nyquist for freq in
                  cutoff_freqs]
111           self.h = firwin(num_taps, cutoff, window='
                  hamming', pass_zero='bandpass')
112       elif filter_type == "Rejeita-Faixa":
113           cutoff = [freq / nyquist for freq in
                  cutoff_freqs]
114           self.h = firwin(num_taps, cutoff, window='
                  hamming', pass_zero='bandstop')
115       else:
116           raise ValueError(f"Tipo de filtro '{
                  filter_type}' n o suportado.")
117
118   def apply_filter(self, input_signal):
119       """
120       Aplica o filtro FIR ao sinal de entrada.
121       """
122       if self.h is None:
123           raise ValueError("Coeficientes do filtro
                  n o definidos.")
124
125       # Converte para float32 se necess rio
126       if input_signal.dtype != np.float32:
127           input_signal = input_signal.astype(np.
                  float32)
128       coeffs = self.h.astype(np.float32)
129
130       if self.use_gpu and self.ctx is not None:
131           # Filtragem GPU usando OpenCL
132           # Prepara os buffers
133           mf = cl.mem_flags
134           try:
135               input_buf = cl.Buffer(self.ctx, mf.
                      READ_ONLY | mf.COPY_HOST_PTR,
                      hostbuf=input_signal)
136               coeff_buf = cl.Buffer(self.ctx, mf.
                      READ_ONLY | mf.COPY_HOST_PTR,
                      hostbuf=coeffs)
137               output_buf = cl.Buffer(self.ctx, mf.
                      WRITE_ONLY, input_signal.nbytes)
138           except cl.MemoryError as e:
139               QMessageBox.critical(None, "Erro OpenCL
                      ", f"Erro ao alocar mem ria no
                      OpenCL: {e}")
140               sys.exit(1)
141
142           # Executa o kernel
143           try:
144               self.program.fir_filter(
145                   self.queue,
146                   (len(input_signal),),
147                   None,
148                   input_buf,
149                   coeff_buf,
150                   output_buf,
151                   np.int32(len(coeffs)),
152                   np.int32(len(input_signal))
153               )
154           except Exception as e:
155               QMessageBox.critical(None, "Erro OpenCL
                      ", f"Erro ao executar kernel
                      OpenCL: {e}")
156               sys.exit(1)
157
158           # L   o resultado
159           output_signal = np.empty_like(input_signal)
160           try:
161               cl.enqueue_copy(self.queue,
                      output_signal, output_buf)
```

```
            except Exception as e:
                QMessageBox.critical(None, "Erro", f"Erro ao copiar dados do OpenCL: {e}")
                sys.exit(1)
            return output_signal
        else:
            # Filtragem CPU usando numpy.convolve
            output_signal = np.convolve(input_signal, coeffs, mode='same')
            return output_signal.astype(np.float32)


class Filter:
    """Classe para representar um filtro individual usando FIRFilter."""
    def __init__(self, filter_type, fs, freq_low=None, freq_high=None, gain=1.0, use_gpu=True):
        self.filter_type = filter_type
        self.freq_low = freq_low
        self.freq_high = freq_high
        self.gain = gain
        self.fs = fs
        self.use_gpu = use_gpu
        self.fir_filter = FIRFilter(num_taps=501, use_gpu=self.use_gpu)
        self.create_filter()

    def create_filter(self):
        if self.filter_type == "Passa-Baixa":
            if self.freq_high is None:
                raise ValueError("Frequência alta deve ser especificada para Passa-Baixa.")
            self.fir_filter.design_filter(
                "Passa-Baixa", [self.freq_high], self.fs)
        elif self.filter_type == "Passa-Alta":
            if self.freq_low is None:
                raise ValueError("Frequência baixa deve ser especificada para Passa-Alta.")
            self.fir_filter.design_filter(
                "Passa-Alta", [self.freq_low], self.fs)
        elif self.filter_type == "Passa-Faixa":
            if self.freq_low is None or self.freq_high is None:
                raise ValueError("Frequências baixa e alta devem ser especificadas para Passa-Faixa.")
            self.fir_filter.design_filter(
                "Passa-Faixa", [self.freq_low, self.freq_high], self.fs)
        elif self.filter_type == "Rejeita-Faixa":
            if self.freq_low is None or self.freq_high is None:
                raise ValueError("Frequências baixa e alta devem ser especificadas para Rejeita-Faixa.")
            self.fir_filter.design_filter(
                "Rejeita-Faixa", [self.freq_low, self.freq_high], self.fs)
        else:
            pass  # Nenhum

    def apply(self, data):
        # Aplica o filtro
        filtered = self.fir_filter.apply_filter(data) * self.gain
        return filtered


class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Aplicação de Filtros FIR e Decomposição de Sinais com FFT Acelerados por GPU e OpenCL")
        self.setGeometry(100, 100, 1600, 900)  # Tamanho inicial

        # Layout principal
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        main_layout = QHBoxLayout()
        central_widget.setLayout(main_layout)

        # Lado esquerdo: Controles de Áudio e Filtros (10%)
        self.control_filter_group = QGroupBox("Controles e Filtros")
        self.control_filter_layout = QVBoxLayout()

        # Seção de Controles de Áudio
        self.audio_controls_group = QGroupBox("Controles de Áudio")
        self.audio_controls_layout = QVBoxLayout()

        self.load_button = QPushButton("Carregar Áudio")
        self.load_button.clicked.connect(self.select_file)
        self.audio_controls_layout.addWidget(self.load_button)

        self.play_button = QPushButton("Play Original")
        self.play_button.clicked.connect(self.play_audio)
        self.play_button.setEnabled(False)
        self.audio_controls_layout.addWidget(self.play_button)

        self.play_filtered_button = QPushButton("Play Filtrado")
        self.play_filtered_button.clicked.connect(self.play_filtered_audio)
        self.play_filtered_button.setEnabled(False)
        self.audio_controls_layout.addWidget(self.play_filtered_button)

        self.stop_button = QPushButton("Stop")
        self.stop_button.clicked.connect(self.stop_audio)
        self.stop_button.setEnabled(False)
        self.audio_controls_layout.addWidget(self.stop_button)

        self.save_button = QPushButton("Salvar Sinal Filtrado")
        self.save_button.clicked.connect(self.save_filtered_audio)
        self.save_button.setEnabled(False)
        self.audio_controls_layout.addWidget(self.save_button)

        # Checkbox para Aceleração GPU
        self.gpu_checkbox = QCheckBox("Usar Aceleração GPU (OpenCL)")
        self.gpu_checkbox.setChecked(True)
        self.audio_controls_layout.addWidget(self.gpu_checkbox)

        # Checkbox para visualizar em dB ou Magnitude
        self.db_checkbox = QCheckBox("Visualizar FFT em dB")
        self.db_checkbox.setChecked(False)
        self.db_checkbox.stateChanged.connect(self.update_plots)  # Atualiza os gráficos ao
```

```
            mudar o estado
267         self.audio_controls_layout.addWidget(self.
                db_checkbox)
268
269         # Bot o para exportar composi  o harm nica
270         self.export_button = QPushButton("Exportar
                Composi  o Harm nica")
271         self.export_button.clicked.connect(self.
                export_harmonic_composition)
272         self.export_button.setEnabled(False)
273         self.audio_controls_layout.addWidget(self.
                export_button)
274
275         self.status_label = QLabel("Status: Pronto")
276         self.audio_controls_layout.addWidget(self.
                status_label)
277
278         # Barra de progresso para pr -processamento
279         self.progress_bar = QProgressBar()
280         self.progress_bar.setValue(0)
281         self.progress_bar.setVisible(False)
282         self.audio_controls_layout.addWidget(self.
                progress_bar)
283
284         # Espa ador para alinhar os elementos no topo
285         self.audio_controls_layout.addSpacerItem(
                QSpacerItem(20, 40, QSizePolicy.Minimum,
                QSizePolicy.Expanding))
286
287         self.audio_controls_group.setLayout(self.
                audio_controls_layout)
288         self.control_filter_layout.addWidget(self.
                audio_controls_group)
289
290         # Se   o de Filtros
291         self.filters_group = QGroupBox("Filtros")
292         self.filters_layout = QVBoxLayout()
293
294         # Tipo de Filtro
295         self.filter_type_combo = QComboBox()
296         self.filter_type_combo.addItems(["Nenhum", "
                Passa-Baixa", "Passa-Alta", "Passa-Faixa",
                 "Rejeita-Faixa"])
297         self.filter_type_combo.currentTextChanged.
                connect(self.update_filter_inputs)
298         self.filters_layout.addWidget(QLabel("Tipo de
                Filtro:"))
299         self.filters_layout.addWidget(self.
                filter_type_combo)
300
301         # Entrada de Frequ ncia Baixa
302         self.freq_low_input = QLineEdit()
303         self.freq_low_input.setPlaceholderText("Ex:
                1000")
304         self.freq_low_input.setEnabled(False)
305         self.filters_layout.addWidget(QLabel("
                Frequ ncia Baixa (Hz):"))
306         self.filters_layout.addWidget(self.
                freq_low_input)
307
308         # Entrada de Frequ ncia Alta
309         self.freq_high_input = QLineEdit()
310         self.freq_high_input.setPlaceholderText("Ex:
                2000")
311         self.freq_high_input.setEnabled(False)
312         self.filters_layout.addWidget(QLabel("
                Frequ ncia Alta (Hz):"))
313         self.filters_layout.addWidget(self.
                freq_high_input)
314
315         # Ganho
316         self.gain_input = QLineEdit("1.0")  # 1.0 para
                nenhuma altera  o
317         self.filters_layout.addWidget(QLabel("Ganho (
                Amplificar >1, Atenuar <1):"))
318         self.filters_layout.addWidget(self.gain_input)
319
320         # Bot o para adicionar filtro
321         self.add_filter_button = QPushButton("Adicionar
                 Filtro")
322         self.add_filter_button.clicked.connect(self.
                add_filter)
323         self.filters_layout.addWidget(self.
                add_filter_button)
324
325         # Lista de Filtros Ativos
326         self.filters_display = QGroupBox("Filtros
                Ativos")
327         self.filters_display_layout = QVBoxLayout()
328         self.filters_display.setLayout(self.
                filters_display_layout)
329         self.filters_layout.addWidget(self.
                filters_display)
330
331         # Espa ador para alinhar os elementos no topo
332         self.filters_layout.addSpacerItem(QSpacerItem
                (20, 40, QSizePolicy.Minimum, QSizePolicy.
                Expanding))
333
334         self.filters_group.setLayout(self.
                filters_layout)
335         self.control_filter_layout.addWidget(self.
                filters_group)
336
337         # Se   o de Monitoramento de Hardware
338         self.hardware_monitor_group = QGroupBox("
                Monitor de Hardware")
339         self.hardware_monitor_layout = QVBoxLayout()
340
341         # Labels para exibir o uso de hardware
342         self.cpu_usage_label = QLabel("Uso da CPU: 0%")
343         self.ram_usage_label = QLabel("Uso da RAM: 0%")
344         self.gpu_usage_label = QLabel("Uso da GPU: 0%")
                # Novo label para uso da GPU
345         self.hardware_monitor_layout.addWidget(self.
                cpu_usage_label)
346         self.hardware_monitor_layout.addWidget(self.
                ram_usage_label)
347         self.hardware_monitor_layout.addWidget(self.
                gpu_usage_label)  # Adiciona o label de
                uso da GPU
348
349         # Espa ador para alinhar os elementos no topo
350         self.hardware_monitor_layout.addSpacerItem(
                QSpacerItem(20, 40, QSizePolicy.Minimum,
                QSizePolicy.Expanding))
351
352         self.hardware_monitor_group.setLayout(self.
                hardware_monitor_layout)
353         self.control_filter_layout.addWidget(self.
                hardware_monitor_group)
354
355         self.control_filter_group.setLayout(self.
                control_filter_layout)
356         main_layout.addWidget(self.control_filter_group
                , 1)  # Fator de estiramento 1 (10%)
357
358         # Lado direito: Logo e Gr ficos (90%)
359         self.main_display_group = QGroupBox("Gr ficos
                e Logo")
360         self.main_display_layout = QVBoxLayout()
361
362         # Logo centralizado no topo: 10% da altura
363         self.logo_label_center = QLabel()
364         self.load_logo_center()
365         self.logo_label_center.setAlignment(Qt.
                AlignCenter)
366         self.logo_label_center.setSizePolicy(
```

```
            QSizePolicy.Expanding, QSizePolicy.Fixed)
        self.main_display_layout.addWidget(self.
            logo_label_center, 0)  # Fator de
            estiramento 0 (altura fixa)

        # Gr ficos: 90% da  rea  central restante
        self.plots_layout = QGridLayout()

        # Gr fico do Sinal Original
        self.original_plot = pg.PlotWidget(title="Sinal
            Original")
        self.original_plot.setLabel('left', "Amplitude"
            )
        self.original_plot.setLabel('bottom', "Tempo",
            units='s')
        self.original_plot.addLegend()
        self.original_curve = self.original_plot.plot(
            pen='c', name="Original")
        self.original_play_line = self.original_plot.
            addLine(x=0, pen=pg.mkPen('g', width=2))
        self.plots_layout.addWidget(self.original_plot,
            0, 0)

        # FFT do Sinal Original
        self.original_fft_plot = pg.PlotWidget(title="
            FFT do Sinal Original")
        self.original_fft_plot.setLabel('left', "
            Magnitude")
        self.original_fft_plot.setLabel('bottom', "
            Frequ ncia", units='Hz')
        self.original_fft_plot.setXRange(0, 20000)
        self.original_fft_plot.addLegend()
        self.original_fft_curve = self.
            original_fft_plot.plot(pen='m', name="FFT
            Original")
        self.plots_layout.addWidget(self.
            original_fft_plot, 1, 0)

        # Gr fico do Sinal Filtrado
        self.filtered_plot = pg.PlotWidget(title="Sinal
            Filtrado")
        self.filtered_plot.setLabel('left', "Amplitude"
            )
        self.filtered_plot.setLabel('bottom', "Tempo",
            units='s')
        self.filtered_plot.addLegend()
        self.filtered_curve = self.filtered_plot.plot(
            pen='y', name="Filtrado")
        self.filtered_play_line = self.filtered_plot.
            addLine(x=0, pen=pg.mkPen('g', width=2))
        self.plots_layout.addWidget(self.filtered_plot,
            2, 0)

        # FFT do Sinal Filtrado
        self.filtered_fft_plot = pg.PlotWidget(title="
            FFT do Sinal Filtrado")
        self.filtered_fft_plot.setLabel('left', "
            Magnitude")
        self.filtered_fft_plot.setLabel('bottom', "
            Frequ ncia", units='Hz')
        self.filtered_fft_plot.setXRange(0, 20000)
        self.filtered_fft_plot.addLegend()
        self.filtered_fft_curve = self.
            filtered_fft_plot.plot(pen='r', name="FFT
            Filtrado")
        self.plots_layout.addWidget(self.
            filtered_fft_plot, 3, 0)

        self.main_display_layout.addLayout(self.
            plots_layout)
        self.main_display_group.setLayout(self.
            main_display_layout)
        main_layout.addWidget(self.main_display_group,
            9)  # Fator de estiramento 9 (90%)



        # Vari veis de  udio
        self.audio_data = None
        self.filtered_data = None
        self.duration = 0
        self.fs = 44100  # Frequ ncia de amostragem
            padr o
        self.playing = False
        self.playing_filtered = False  # Controle para
            reprodu  o filtrada

        # Filtros ativos
        self.active_filters = []

        # Timer para atualiza  es em tempo real
        self.timer = QTimer()
        self.timer.setInterval(10)  # Atualiza a cada
            100 ms
        self.timer.timeout.connect(self.
            update_realtime_plots_timer)

        # Timer para monitoramento de hardware
        self.hardware_timer = QTimer()
        self.hardware_timer.setInterval(1000)  #
            Atualiza a cada segundo
        self.hardware_timer.timeout.connect(self.
            update_hardware_usage)
        self.hardware_timer.start()

        # Vari veis para rastrear o tempo de
            reprodu  o
        self.play_start_time = None
        self.filtered_play_start_time = None

        # Arquivos tempor rios para reprodu  o
        self.temp_original_wav = "temp_original.wav"
        self.temp_filtered_wav = "temp_filtered.wav"

        # Dados pr -computados para reprodu  o
        self.precomputed_original_fft = None
        self.precomputed_filtered_fft = None

        # Amplitudes m ximas pr -computadas para
            escala do eixo Y
        self.original_max_amplitude = 1000.0
        self.filtered_max_amplitude = 1000.0

        # Eixo de tempo pr -computado para o sinal
            original
        self.times = None

        # Composi  o harm nica pr -computada
        self.harmonic_composition = None

    def resizeEvent(self, event):
        """Atualiza os tamanhos fixos dos grupos com
            base na nova dimens o da janela."""
        new_width = self.width()
        new_height = self.height()

        # Atualiza a altura do logo para 10% da altura
            da janela
        self.logo_label_center.setFixedHeight(int(
            new_height * 0.15))
        if hasattr(self, 'logo_pixmap'):
            self.logo_label_center.setPixmap(
                self.logo_pixmap.scaled(
                    self.logo_label_center.width(),
                    self.logo_label_center.height(),
                    Qt.KeepAspectRatio,
                    Qt.SmoothTransformation
                )
            )
```

```
473     def load_logo_center(self):
474         """Carrega o logo da pasta 'assets' com
                prioridade png, jpeg, jpg."""
475         logo_dir = "assets"
476         logo_name = "logo"
477         extensions = ['.png', '.jpeg', '.jpg']
478 
479         logo_path = None
480         for ext in extensions:
481             path = os.path.join(logo_dir, logo_name +
                    ext)
482             if os.path.exists(path):
483                 logo_path = path
484                 break
485 
486         if logo_path:
487             self.logo_pixmap = pg.QtGui.QPixmap(
                    logo_path)
488             self.logo_label_center.setPixmap(
489                 self.logo_pixmap.scaled(
490                     self.logo_label_center.width(),
491                     self.logo_label_center.height(),
492                     Qt.KeepAspectRatio,
493                     Qt.SmoothTransformation
494                 )
495             )
496         else:
497             self.logo_label_center.setText("Logo n o
                    encontrado")
498 
499     def update_filter_inputs(self, filter_type):
500         """Habilita ou desabilita campos de frequ ncia
                com base no tipo de filtro selecionado.
               """
501         if filter_type == "Passa-Baixa":
502             self.freq_high_input.setEnabled(True)
503             self.freq_low_input.setEnabled(False)
504         elif filter_type == "Passa-Alta":
505             self.freq_low_input.setEnabled(True)
506             self.freq_high_input.setEnabled(False)
507         elif filter_type in ["Passa-Faixa", "Rejeita-
                Faixa"]:
508             self.freq_low_input.setEnabled(True)
509             self.freq_high_input.setEnabled(True)
510         else:  # Nenhum
511             self.freq_low_input.setEnabled(False)
512             self.freq_high_input.setEnabled(False)
513 
514     def add_filter(self):
515         filter_type = self.filter_type_combo.
                currentText()
516         if filter_type == "Nenhum":
517             QMessageBox.warning(self, "Aviso", "
                    Selecione um tipo de filtro diferente
                    de 'Nenhum'.")
518             return
519 
520         try:
521             freq_low_text = self.freq_low_input.text()
522             freq_high_text = self.freq_high_input.text
                    ()
523             gain = float(self.gain_input.text())
524 
525             freq_low = float(freq_low_text) if
                    freq_low_text and self.freq_low_input.
                    isEnabled() else None
526             freq_high = float(freq_high_text) if
                    freq_high_text and self.
                    freq_high_input.isEnabled() else None
527         except ValueError:
528             QMessageBox.warning(self, "Erro", "Por
                    favor, insira valores num ricos
                    v lidos para as frequ ncias e ganho.
                    ")
529             return
530 
531         # Valida  o de frequ ncia com base no tipo
                de filtro
532         if filter_type == "Passa-Baixa":
533             if freq_high is None:
534                 QMessageBox.warning(self, "Erro", "
                        Frequ ncia alta deve ser
                        especificada para Passa-Baixa.")
535                 return
536             if not (0 < freq_high < self.fs / 2):
537                 QMessageBox.warning(self, "Erro", f"A
                        frequ ncia alta deve estar entre
                        0 e {self.fs / 2} Hz.")
538                 return
539         elif filter_type == "Passa-Alta":
540             if freq_low is None:
541                 QMessageBox.warning(self, "Erro", "
                        Frequ ncia baixa deve ser
                        especificada para Passa-Alta.")
542                 return
543             if not (0 < freq_low < self.fs / 2):
544                 QMessageBox.warning(self, "Erro", f"A
                        frequ ncia baixa deve estar entre
                         0 e {self.fs / 2} Hz.")
545                 return
546         elif filter_type in ["Passa-Faixa", "Rejeita-
                Faixa"]:
547             if freq_low is None or freq_high is None:
548                 QMessageBox.warning(self, "Erro", f"
                        Frequ ncias baixa e alta devem
                        ser especificadas para {
                        filter_type}.")
549                 return
550             if not (0 < freq_low < freq_high < self.fs
                     / 2):
551                 QMessageBox.warning(self, "Erro", f"As
                        frequ ncias devem estar entre 0 e
                         {self.fs / 2} Hz, e a frequ ncia
                         baixa deve ser menor que a alta."
                        )
552                 return
553 
554         use_gpu = self.gpu_checkbox.isChecked()
555         new_filter = Filter(filter_type, self.fs,
                freq_low, freq_high, gain, use_gpu)
556         self.active_filters.append(new_filter)
557 
558         # Atualiza a interface para mostrar filtros
                ativos com op   o de remo   o
559         filter_item = QWidget()
560         filter_layout = QHBoxLayout()
561         filter_label = QLabel()
562         if filter_type == "Passa-Baixa":
563             filter_label.setText(f"{filter_type}: < {
                    freq_high} Hz | Ganho: {gain}")
564         elif filter_type == "Passa-Alta":
565             filter_label.setText(f"{filter_type}: > {
                    freq_low} Hz | Ganho: {gain}")
566         elif filter_type == "Passa-Faixa":
567             filter_label.setText(f"{filter_type}: {
                    freq_low} - {freq_high} Hz | Ganho: {
                    gain}")
568         elif filter_type == "Rejeita-Faixa":
569             filter_label.setText(f"{filter_type}: {
                    freq_low} - {freq_high} Hz | Ganho: {
                    gain}")
570 
571         remove_button = QPushButton("Remover")
572         remove_button.clicked.connect(lambda _, f=
                new_filter, w=filter_item: self.
                remove_filter(f, w))
573         filter_layout.addWidget(filter_label)
574         filter_layout.addWidget(remove_button)
```

```
        filter_item.setLayout(filter_layout)
        self.filters_display_layout.addWidget(
            filter_item)

        # Atualiza os gr ficos  ap s  adicionar o
            filtro
        if self.audio_data is not None:
            self.preprocess_audio()

    def remove_filter(self, filt, widget):
        if filt in self.active_filters:
            self.active_filters.remove(filt)
            self.filters_display_layout.removeWidget(
                widget)
            widget.deleteLater()

            # Atualiza os gr ficos  ap s  remover o
                filtro
            if self.audio_data is not None:
                self.preprocess_audio()

    def load_audio(self, file_path):
        try:
            # Carrega o  udio  usando soundfile
            data, self.fs = sf.read(file_path)
            self.audio_data = data.astype(np.float32)

            # Verifica o n mero de canais
            if len(self.audio_data.shape) == 2:
                print(" udio  est reo  carregado.")
                # Converte para mono pela m dia dos
                    canais
                self.audio_data = self.audio_data.mean(
                    axis=1)
                print(f"Forma  ap s  convers o para
                    mono: {self.audio_data.shape}")
            else:
                print(" udio  mono  carregado.")

            self.duration = len(self.audio_data) / self
                .fs

            # Normaliza o  udio
            if np.max(np.abs(self.audio_data)) != 0:
                self.audio_data /= np.max(np.abs(self.
                    audio_data))

            # Pr -computa o eixo de tempo
            self.times = np.linspace(0, self.duration,
                num=len(self.audio_data))

            # Pr -processa o  udio  (filtragem e FFT)
            self.preprocess_audio()

            # Habilita os bot es de salvar e
                reproduzir filtrado
            self.save_button.setEnabled(True)
            self.play_button.setEnabled(True)
            self.play_filtered_button.setEnabled(True)
            self.export_button.setEnabled(True)
        except Exception as e:
            print(f"Falha ao carregar o  udio : {e}")
            QMessageBox.critical(self, "Erro", f"Falha
                ao carregar o  udio : {e}")

    def preprocess_audio(self):
        """Pr -processa os dados de  udio : aplica
            filtros e computa FFTs."""
        self.status_label.setText("Status: Pr -
            processando...")
        self.progress_bar.setVisible(True)
        self.progress_bar.setValue(0)
        QApplication.processEvents()

        # Aplica filtros
        try:
            self.filtered_data = self.apply_filters(
                self.audio_data)
        except Exception as e:
            QMessageBox.critical(self, "Erro", f"Erro
                ao aplicar filtros: {e}")
            self.progress_bar.setVisible(False)
            self.status_label.setText("Status: Erro")
            return

        # Atualiza o progresso
        self.progress_bar.setValue(50)
        QApplication.processEvents()

        # Computa FFTs
        self.precomputed_original_fft = self.
            compute_fft(self.audio_data)
        self.precomputed_filtered_fft = self.
            compute_fft(self.filtered_data)

        # Pr -computa amplitudes m ximas para escala
            do eixo Y
        self.original_max_amplitude = np.max(np.abs(
            self.audio_data))
        self.filtered_max_amplitude = np.max(np.abs(
            self.filtered_data))

        # Pr -computa a composi  o harm nica
        self.compute_harmonic_composition()

        # Atualiza o progresso
        self.progress_bar.setValue(100)
        QApplication.processEvents()

        # Atualiza os gr ficos
        self.update_plots()

        # Oculta a barra de progresso
        self.progress_bar.setVisible(False)
        self.status_label.setText("Status: Pronto")

    def compute_harmonic_composition(self):
        """Computa a composi  o harm nica dos sinais
            original e filtrado."""
        freqs, magnitude = self.
            precomputed_filtered_fft
        if self.db_checkbox.isChecked():
            magnitude_db = 20 * np.log10(magnitude + 1e
                -12)  # Converte magnitude para dB
            self.harmonic_composition = pd.DataFrame({'
                Frequency (Hz)': freqs, 'Magnitude (dB
                )': magnitude_db})
        else:
            self.harmonic_composition = pd.DataFrame({'
                Frequency (Hz)': freqs, 'Magnitude':
                magnitude})

    def compute_fft(self, data_chunk):
        """Computa a FFT do chunk de dados fornecido.
            """
        n = len(data_chunk)
        fft_result = np.fft.rfft(data_chunk)
        freqs = np.fft.rfftfreq(n, 1.0 / self.fs)
        magnitude = np.abs(fft_result)
        return freqs, magnitude

    def apply_filters(self, data):
        filtered = np.copy(data)
        for filt in self.active_filters:
            filtered = filt.apply(filtered)
        return filtered

    def update_plots(self):
```

```
        if self.audio_data is None:
            return

        # Verifica se o usu rio deseja visualizar em dB
        use_db = self.db_checkbox.isChecked()

        # Original Signal
        self.original_curve.setData(self.times, self.audio_data)
        self.original_plot.setYRange(-self.original_max_amplitude, self.original_max_amplitude)
        self.original_plot.setXRange(0, self.duration)

        # FFT of Original Signal
        freqs, magnitude = self.precomputed_original_fft
        if use_db:
            magnitude = 20 * np.log10(magnitude + 1e-12)
            self.original_fft_plot.setLabel('left', "Magnitude (dB)")
        else:
            self.original_fft_plot.setLabel('left', "Magnitude")
        self.original_fft_curve.setData(freqs, magnitude)

        # Filtered Signal
        self.filtered_curve.setData(self.times, self.filtered_data)
        self.filtered_plot.setYRange(-self.filtered_max_amplitude, self.filtered_max_amplitude)
        self.filtered_plot.setXRange(0, self.duration)

        # FFT of Filtered Signal
        freqs_f, magnitude_f = self.precomputed_filtered_fft
        if use_db:
            magnitude_f = 20 * np.log10(magnitude_f + 1e-12)
            self.filtered_fft_plot.setLabel('left', "Magnitude (dB)")
        else:
            self.filtered_fft_plot.setLabel('left', "Magnitude")
        self.filtered_fft_curve.setData(freqs_f, magnitude_f)

    def update_realtime_plots_timer(self):
        current_time = time.time()
        if self.playing:
            elapsed_time = current_time - self.play_start_time
            current_sample = int(elapsed_time * self.fs)
            if current_sample >= len(self.audio_data):
                self.timer.stop()
                self.playing = False
                self.play_button.setEnabled(True)
                self.play_filtered_button.setEnabled(True)
                self.stop_button.setEnabled(False)
                self.status_label.setText("Status: Reprodu  o Finalizada")
                return

            # Atualiza a posi  o da linha de reprodu  o
            self.original_play_line.setValue(elapsed_time)

        elif self.playing_filtered:
            elapsed_time = current_time - self.filtered_play_start_time
            current_sample = int(elapsed_time * self.fs)
            if current_sample >= len(self.filtered_data):
                self.timer.stop()
                self.playing_filtered = False
                self.play_button.setEnabled(True)
                self.play_filtered_button.setEnabled(True)
                self.stop_button.setEnabled(False)
                self.status_label.setText("Status: Reprodu  o Finalizada")
                return

            # Atualiza a posi  o da linha de reprodu  o
            self.filtered_play_line.setValue(elapsed_time)

            # Atualiza a janela do sinal filtrado
            window_duration = 1.0  # segundos
            window_size = int(self.fs * window_duration)
            start_idx = max(0, current_sample - window_size)
            times = self.times[start_idx:current_sample]
            self.filtered_curve.setData(times, self.filtered_data[start_idx:current_sample])
            self.filtered_plot.setYRange(-self.filtered_max_amplitude, self.filtered_max_amplitude)
            self.filtered_plot.setXRange(elapsed_time - window_duration, elapsed_time)

            # Computa a FFT instant nea
            data_window = self.filtered_data[start_idx:current_sample]
            freqs_f, magnitude_f = self.compute_fft(data_window)
            if self.db_checkbox.isChecked():
                magnitude_f = 20 * np.log10(magnitude_f + 1e-12)
                self.filtered_fft_plot.setLabel('left', "Magnitude (dB)")
            else:
                self.filtered_fft_plot.setLabel('left', "Magnitude")
            self.filtered_fft_curve.setData(freqs_f, magnitude_f)

    def update_hardware_usage(self):
        # Atualiza o uso da CPU
        cpu_usage = psutil.cpu_percent(interval=None)
        self.cpu_usage_label.setText(f"Uso da CPU: {cpu_usage}%")

        # Atualiza o uso da RAM
        ram_usage = psutil.virtual_memory().percent
        self.ram_usage_label.setText(f"Uso da RAM: {ram_usage}%")

        # Atualiza o uso da GPU
        try:
            gpus = GPUtil.getGPUs()
            if gpus:
                gpu = gpus[0]
                gpu_usage = gpu.load * 100
                self.gpu_usage_label.setText(f"Uso da GPU: {gpu_usage:.1f}%")
```

```
            else:
                self.gpu_usage_label.setText("Uso da GPU: N/A")
        except Exception as e:
            self.gpu_usage_label.setText("Uso da GPU: Erro")

    def play_audio(self):
        if self.audio_data is None:
            QMessageBox.warning(self, "Aviso", "Por favor, carregue um arquivo de udio primeiro.")
            return

        if self.playing:
            return

        # Salva o udio original em um arquivo tempor rio para reprodu o via mixer.music
        try:
            # Normaliza e salva em WAV tempor rio
            if np.max(np.abs(self.audio_data)) != 0:
                normalized_data = self.audio_data / np.max(np.abs(self.audio_data))
            else:
                normalized_data = self.audio_data

            sf.write(self.temp_original_wav, normalized_data, self.fs)
            pygame.mixer.music.load(self.temp_original_wav)
            pygame.mixer.music.play()
            self.playing = True
            self.play_button.setEnabled(False)
            self.play_filtered_button.setEnabled(False)
            self.stop_button.setEnabled(True)
            self.status_label.setText("Status: Reproduzindo Original...")

            self.play_start_time = time.time()
            self.timer.start()
        except Exception as e:
            QMessageBox.critical(self, "Erro", f"Falha ao reproduzir o udio original: {e}")

    def play_filtered_audio(self):
        if self.filtered_data is None:
            QMessageBox.warning(self, "Aviso", "N o h sinal filtrado para reproduzir.")
            return

        if self.playing_filtered:
            return

        # Converte o sinal filtrado para int16 para reprodu o
        if np.max(np.abs(self.filtered_data)) != 0:
            normalized_data = self.filtered_data / np.max(np.abs(self.filtered_data))
        else:
            normalized_data = self.filtered_data

        # Salva o udio filtrado em um arquivo tempor rio para reprodu o via mixer.Sound
        try:
            sf.write(self.temp_filtered_wav, normalized_data, self.fs)
            sound_filtered = pygame.mixer.Sound(self.temp_filtered_wav)
            self.filtered_channel = pygame.mixer.Channel(1)  # Usa o canal 1 para o sinal filtrado
            self.filtered_channel.play(sound_filtered)
            self.playing_filtered = True
            self.play_filtered_button.setEnabled(False)
            self.play_button.setEnabled(False)
            self.stop_button.setEnabled(True)
            self.status_label.setText("Status: Reproduzindo Filtrado...")

            self.filtered_play_start_time = time.time()
            self.timer.start()
        except Exception as e:
            QMessageBox.critical(self, "Erro", f"Falha ao reproduzir o udio filtrado: {e}")

    def stop_audio(self):
        if self.playing:
            pygame.mixer.music.stop()
            self.playing = False
            self.play_button.setEnabled(True)
            self.play_filtered_button.setEnabled(True)
            self.stop_button.setEnabled(False)
            self.status_label.setText("Status: Parado")
            self.timer.stop()

            # Remove arquivo tempor rio
            if os.path.exists(self.temp_original_wav):
                os.remove(self.temp_original_wav)

        if self.playing_filtered:
            self.filtered_channel.stop()
            self.playing_filtered = False
            self.play_filtered_button.setEnabled(True)
            self.play_button.setEnabled(True)
            self.stop_button.setEnabled(False)
            self.status_label.setText("Status: Parado")
            self.timer.stop()

            # Remove arquivo tempor rio
            if os.path.exists(self.temp_filtered_wav):
                os.remove(self.temp_filtered_wav)

    def save_filtered_audio(self):
        if self.filtered_data is None:
            QMessageBox.warning(self, "Aviso", "N o h sinal filtrado para salvar.")
            return

        # Seleciona o formato de salvamento
        format_dialog = QMessageBox()
        format_dialog.setWindowTitle("Selecionar Formato")
        format_dialog.setText("Escolha o formato para salvar o udio filtrado:")
        wav_button = format_dialog.addButton("WAV", QMessageBox.AcceptRole)
        mp3_button = format_dialog.addButton("MP3", QMessageBox.AcceptRole)
        cancel_button = format_dialog.addButton("Cancelar", QMessageBox.RejectRole)
        format_dialog.exec_()

        if format_dialog.clickedButton() == wav_button:
            selected_format = "WAV"
            file_filter = "Arquivo WAV (*.wav)"
        elif format_dialog.clickedButton() == mp3_button:
            selected_format = "MP3"
            file_filter = "Arquivo MP3 (*.mp3)"
        else:
            return  # Cancelado

        save_path, _ = QFileDialog.getSaveFileName(
            self, "Salvar Sinal Filtrado", "", file_filter
```

```
        )
        if save_path:
            try:
                # Normaliza o sinal filtrado para evitar clipping
                if np.max(np.abs(self.filtered_data)) != 0:
                    normalized_data = self.filtered_data / np.max(np.abs(self.filtered_data))
                else:
                    normalized_data = self.filtered_data

                print(f"M ximo absoluto ap s normaliza  o: {np.max(np.abs(normalized_data))}")  # Debug

                if selected_format == "WAV":
                    # Salva usando soundfile
                    sf.write(save_path, normalized_data, self.fs)
                elif selected_format == "MP3":
                    # Salva temporariamente em WAV e converte para MP3 usando pydub
                    temp_wav = save_path + ".temp.wav"
                    sf.write(temp_wav, normalized_data, self.fs)
                    sound = AudioSegment.from_wav(temp_wav)
                    sound.export(save_path, format="mp3")
                    os.remove(temp_wav)  # Remove arquivo tempor rio
                QMessageBox.information(self, "Sucesso", f"Sinal filtrado salvo em {save_path}")

                # Salva os par metros do filtro e composi  o harm nica
                params_save_path = save_path + "_params.txt"
                with open(params_save_path, 'w') as f:
                    f.write("Par metros dos Filtros Aplicados:\n")
                    for filt in self.active_filters:
                        f.write(f"Filtro: {filt.filter_type}, Freq Low: {filt.freq_low}, Freq High: {filt.freq_high}, Ganho: {filt.gain}\n")
                QMessageBox.information(self, "Sucesso", f"Par metros salvos em {params_save_path}")

            except Exception as e:
                QMessageBox.critical(self, "Erro", f"Falha ao salvar o  udio  filtrado: {e}")

    def select_file(self):
        file_path, _ = QFileDialog.getOpenFileName(
            self, "Selecionar Arquivo de  udio ", "", "Arquivos de  udio  (*.wav *.mp3)"
        )
        if file_path:
            self.status_label.setText("Status: Carregando...")
            QApplication.processEvents()
            try:
                self.load_audio(file_path)
                self.status_label.setText("Status: Arquivo Carregado")
                self.play_button.setEnabled(True)
                self.play_filtered_button.setEnabled(True)
            except Exception as e:
                QMessageBox.critical(self, "Erro", f"Falha ao carregar o  udio : {e}")

    def export_harmonic_composition(self):
        if self.harmonic_composition is None:
            QMessageBox.warning(self, "Aviso", "N o h  composi  o harm nica para exportar.")
            return

        save_path, _ = QFileDialog.getSaveFileName(
            self, "Exportar Composi  o Harm nica", "", "Arquivo CSV (*.csv)"
        )
        if save_path:
            try:
                self.harmonic_composition.to_csv(save_path, index=False)
                QMessageBox.information(self, "Sucesso", f"Composi  o harm nica exportada para {save_path}")
            except Exception as e:
                QMessageBox.critical(self, "Erro", f"Falha ao exportar a composi  o harm nica: {e}")


def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())


if __name__ == "__main__":
    main()
```

Listing 1: Código completo da aplicação `FFTsoundAnalyzer`