NEURALYZER: AN AI CHATBOT IN PARTIELS

Pierre GUILLOT (pierre.guillot@ircam.fr) [1]

[1] *IMR*, *STMS UMR 9912*, **IRCAM - Centre Pompidou**, Paris, France

ABSTRACT

This paper introduces Neuralyzer, an extension to the sound analysis software Partiels developed at IRCAM. Implemented as a conversational agent coupled with a server based on the Model Context Protocol (MCP), Neuralyzer assists users in the analysis of digital audio files by facilitating parameter configuration and automating repetitive tasks. We investigate how large language models (LLMs) can introduce an intelligent assistance layer within a sound analysis environment. Particular attention is given to local inference, motivated by constraints related to accessibility, data privacy, and computational efficiency. The text retraces the technical evolution of the project, from an initial phase based on remote APIs to the integration of locally executed models and adoption of MCP, enabling structured and controlled interaction between the model and the software's internal features without compromising application stability. The different architectures tested are critically analyzed, and initial technical and functional results are presented. Overall, this work suggests that a controlled integration of language models can make sound exploration more accessible in research, pedagogy, and musical creation contexts. These features also make it possible to use them in place of a scripting language to perform automation.

1. CONTEXT AND CHALLENGES OF NEURALYZER

Neuralyzer is a set of features integrated into the Partiels [1] software [1] to assist users with interpretation, configuration, and chaining of analyses. Neuralyzer relies on large language model (LLM) inference and the use of chatbots [2, 3]. It is not a standalone software application, but a functional extension of the existing environment. To clearly frame the scientific and technical stakes and clarify the issues Neuralyzer aims to address, we briefly recall the architecture and evolution of the Partiels project, especially its recent developments.

1.1 Overview of Partiels

Partiels is a Vamp plug-in host, a format originally designed at the Centre for Digital Music (Queen Mary University of London) for digital audio stream analysis [4]. The software architecture is based on a modular model: analyses are implemented as dynamically loaded plug-ins. This approach enables the integration of analysis engines developed internally or by third-party institutions, the addition of new analyses without modifying the application core, and the coexistence of heterogeneous models (algorithmic and neural). As detailed in previous publications [1], although the Vamp SDK provides a robust generic framework, it presents several structural limitations: no native ability to chain analyses, non-audio inputs are restricted to scalar values, and outputs are restricted to simple types (label, scalar, vector).

These constraints become problematic when engines require inter-analysis dependencies, for example when the results of one analysis (a set of scalars or labels) must be used as input to another analysis (transcription followed by alignment), or when outputs are multidimensional, i.e., when a result frame contains several scalars representing distinct data (such as the description of a partial represented by a frequency, amplitude, phase, and confidence score). These limitations were addressed through the IRCAM Vamp Extension (IVE), a programming interface specifically developed to extend Vamp plug-in input and output mechanisms. In particular, IVE enables analysis chaining through a more flexible input system and supports transmission of enriched results.

1.2 Evolution of Partiels

Development of Partiels began in 2020. Since its first publication in 2021, followed by free availability in 2023 and the move to GPLv3 open-source in 2024, the application has undergone several structural evolutions. Partiels currently offers the following features:

- an extensive set of analyses, which includes many tools historically available in AudioSculpt [5, 6];
- new analyses based on neural models, such as Whisper [7] and VAX [8];
- an architecture and data model ensuring backward compatibility of documents, allowing integration of new features and modification of existing tools without breaking continuity;
- a structured organization into documents, groups, and analysis tracks, facilitating comparison and superposition of results;

[1] https://github.com/Ircam-Partiels/Partiels

Copyright: © 2026. This is an open-access article distributed under the terms of the Creative Commons Attribution 3.0 Unported License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

- an advanced graphical properties system for modifying result representations (colors, fonts, positioning, etc.);
- extended interoperability through export to various formats (CSV, JSON, CUE, SDIF, PNG, JPEG, etc.) and through the Open Sound Control (OSC) protocol [9].

Recent developments also introduced a preset system to facilitate adaptation of analyses to specific use contexts, a translation system encouraging community contributions, and improvement of export mechanisms to environments such as Max, [2] Pure Data [10], and Reaper. [3]

At this stage, Partiels can be considered a tool with a high level of functional maturity, used in composition, teaching, and research contexts.

1.3 Functional and Technical Bottlenecks

Despite this maturity, several technical and functional bottlenecks remain. Current challenges for Partiels include expanding accessibility for less expert users, improving interfaces, and integrating new interaction layers. Two recurrent use cases illustrate these limitations: configuring the discrete Fourier transform and analyzing multiple harmonic partials.

1.3.1 Configuring the Discrete Fourier Transform (DFT)

The discrete Fourier transform (DFT) provides a frequency-domain representation of the audio signal and underpins many analyses such as spectral centroid, fundamental frequency estimation, and spectrogram generation. Choosing window size, windowing type, window overlapping, and oversampling requires a precise understanding of the underlying mechanisms. The technical vocabulary involved does not necessarily match all users' conceptual frameworks. Certain parameters may be reinterpreted in terms of frequency and temporal resolution to enhance intelligibility for non-expert users. Others could be described with more directly interpretable vocabulary, such as zero-padding in the case of oversampling.

1.3.2 Analysis of Multiple Harmonic Partials

Harmonic partial analyses from engines such as PM2 [11] produce multidimensional data (frequency, amplitude, phase) for each partial. To provide an adequate representation of results, each partial is associated with a distinct analysis track. All tracks share the same parameters except for the partial index. In addition, to obtain robust results, these analyses require fundamental frequency as input. Performing an analysis of twenty harmonic partials therefore requires several steps: creating a group, running an initial fundamental-frequency analysis, creating twenty partial analyses, adjusting the partial index for each track (and, where possible, renaming the tracks accordingly), and finally linking the fundamental-frequency analysis as input to all partial analyses. This is a set of repetitive and time-consuming operations that can lead to configuration errors.

1.3.3 Challenges in User Workflow and Accessibility

These two situations raise two questions:

- How can analyses requiring a high level of technical expertise or usage-dependent vocabulary be parameterized?
- How can the creation of multiple analyses be automated while varying only selected parameters?

Neuralyzer was developed in this context, not as a replacement for existing mechanisms, but as an assistance layer intended to reduce cognitive load and redundancies associated with creating and configuring analyses.

2. NEURALYZER DESIGN SPECIFICATIONS AND PROTOTYPING

Beyond strictly functional issues, this project serves as a critical experimentation ground for different approaches to integrating large language models (LLMs). The choice of integration strategy is shaped by a set of ethical, economic, and ecological constraints specific to Partiels' application domains. Free access is a fundamental requirement given the educational, academic, and research contexts in which Partiels is used. In these contexts, it is not acceptable to require students or users to pay per request to access integrated assistance. Protection of personal data is also a central concern, particularly under French data protection regulations. Furthermore, environmental impact constitutes an important factor in designing an efficient solution that minimizes unnecessary computational resources and network traffic.

In addition to these constraints, the project pursues several scientific and technical objectives. A primary goal is to explore LLM integration within an open-source application framework, allowing users to compare different approaches, models, and inference parameters in real usage contexts. Beyond the quality of generated responses, this comparative capability is essential for both research and pedagogical purposes. Several integration strategies were considered and tested. The selected choices reflect a balance between technical constraints related to the development and distribution of Partiels, ergonomic and user-experience considerations, and the scientific and technological interest of specific solutions. Although presented here in a logical integration order, some approaches were explored without immediate integration plans, while others were developed from the earliest project phases.

2.1 Remote API Approach

A first approach relies on existing large language model services such as ChatGPT (OpenAI) [12] or Le Chat (Mistral). [4] This approach does not require in-depth

[2] `https://cycling74.com/products/max`
[3] `https://www.reaper.fm/`
[4] `https://mistral.ai/`

expertise in LLM inference: users provide a personal API key, allowing Partiels to send prompts and display responses within an integrated interface, with explicit role distinction.

However, these models have only partial knowledge of Partiels and lack operational understanding of its internal structure. Effective interaction therefore requires supplying explicit context, including documentation and a structured description of the current document state. To address this, Partiels provides introspection mechanisms that export the current document state in JSON or XML format, which the model can then modify to propose updates.

Although this approach benefits from high-performance external services and standardized APIs, it presents several limitations. Costs can become significant due to large prompt sizes, especially given the richness of document representations. Remote inference also raises ecological concerns and data protection issues, depending on service policies.

2.2 Local Model Inference

A more economically and ecologically sustainable approach is to perform inference locally. The project adopts the llama.cpp library, an open-source C++ framework developed by Gerganov and distributed under the MIT license. This choice is motivated by several factors: active maintenance, a mature GGML-based ecosystem, [5] cross-platform compatibility (notably Windows, macOS, and Linux), and the ability to integrate directly into Partiels without requiring external dependencies. It also provides access to a wide range of open models (e.g., Mistral [13], Qwen [14], LLaMA [15], Phi [16]) and fine control over inference parameters.

Neuralyzer includes a chat interface where users can select the model and adjust key parameters: context size, [6] batch size, [7] minimum probability, [8] and temperature. [9] Partiels provides several preconfigured Ministral 3 [10] and Qwen 3.5 [17] models (various sizes and quantizations), with support for manual installation. Conversation formatting relies by default on the template embedded in the GGUF model, with the option for user-defined templates. [11]

At initialization, Neuralyzer sends a system prompt defining its role, available plug-ins, and expected output format (`<response>` and `<document>` tags). For each request, the current document state is serialized to XML and injected into the context. The model's output is then parsed: `<response>` content is displayed, while `<document>` content is interpreted to update the document. An excerpt from this system query is shown below:

```
Your goal is to assist users in analyzing
audio files..
Available Plugins:
...
Return your response strictly following
this format:
<response>
[Your text in natural language]
</response>
<document>
[The XML description of the modifications]
</document>
```

This approach ensures free usage, full data control, and no reliance on remote servers, while remaining relatively simple to integrate into the existing architecture. Although local inference is feasible on personal machines, performance remains limited: models often fail to strictly respect the required syntax, leading to potential errors in document modification.

2.3 Model Context Protocol (MCP) Integration

A Partiels document can be converted to a textual representation (XML or JSON), and vice versa. This capability is notably used when saving and loading files. The textual representation contains all information needed for complete document restoration, including unique track and group identifiers as well as many parameters and metadata items. This complex hierarchical structure is difficult for models to manipulate reliably.

The proposed solution therefore avoids exposing the full document representation and instead relies on a set of clearly typed methods that modify specific elements. Each method has an explicit input schema (signature), making it possible to detect errors and prevent inconsistent modifications. The Model Context Protocol (MCP), [12] introduced in late 2024 by Anthropic, allows description of such methods (tools) together with their input and output schemas. Recent models (Claude, Qwen, GPT, Mistral) include support for this protocol.

Recent work has begun exploring MCP applications in the audio domain. Fan [18] introduced MCP2OSC, a server enabling parametric control of Open Sound Control (OSC) compatible audio systems through natural language prompts, demonstrating how Claude can generate OSC messages and manage parameter schemas. The AudioFab framework [19] leverages MCP as a foundational protocol for tool management in an intelligent audio processing ecosystem, supporting several audio techniques across speech, sound, and music through a modular architecture of containerized tools. Orlarey and Letz recently demonstrated how, using MCP, an LLM can generate FAUST programs, as well as analyze and control them [20]. Many projects allow users to create MCP servers to control musical environments, such as SuperCollider, [13]

[5] https://github.com/ggerganov/ggml

[6] Maximum number of tokens the model can process simultaneously.

[7] Number of tokens processed in parallel during request handling.

[8] Threshold below which model predictions are discarded, ensuring that only results with sufficient confidence are retained

[9] Parameter that controls the randomness of a model's output by scaling probability distributions, with higher values producing more diverse and less deterministic results.

[10] https://huggingface.co/collections/mistralai/ministral-3

[11] The conversion template allows you to distinguish between different roles, such as user, assistant, and system.

[12] https://modelcontextprotocol.io

[13] https://github.com/Tok/SuperColliderMCP

Reaper, [14] Max, [15] and PureData. [16] This project is being developed as part of the same initiative but was designed primarily for local use, although it is compatible with commercial online services.

In Neuralyzer, MCP support is implemented through a dispatcher class communicating in JSON format. This class serves two primary functions: it exposes the formal description of available methods (including name, description, and input/output schemas) and translates model-invoked method calls into Partiels-specific internal actions. The MCP tools enable the control and modification of plug-ins, document structure, audio files, groups, and analysis tracks. The methods offered by Neuralyzer via MCP cover the following areas:

- Plug-ins: List installed Vamp plug-ins (identifiers and descriptions).
- Document: Get and set the overall status (groups, tracks), the layout of audio files and their audio channels, and the zoom.
- Audio files: Get metadata (format, sample rate, duration, channels).
- Groups: Get and set track layout; create and delete groups.
- Tracks: Get and set type, name, description, parameters, inputs, results, and graphical properties; analysis parameters; create tracks from plug-ins and delete tracks.

To test this interface, Neuralyzer can communicate with Claude Desktop [17] through an MCP server integrated into Partiels. Initial results proved very encouraging: Claude, developed by Anthropic, provides robust protocol support and is able to describe and modify the document from method descriptions.

2.4 Neuralyzer's Overall System

The next step was to integrate the MCP protocol into the local inference engine. The llama.cpp [18] library provides interfaces that make it easier to declare methods and interpret calls made by compatible models. Interactions with Neuralyzer occur in sequences of successive requests. During this process, the model must maintain explicit consistency with the current document state, and all accessible operations are handled through methods exposed via MCP.

Upon receiving a request, the model formulates an action strategy: either responding directly to the user or invoking one or more MCP methods to modify the document (see Fig. 1). Each call is validated by the internal dispatcher, which returns either an error description or a structured report of the changes made.

[14] https://github.com/dschuler36/reaper-mcp-server

[15] https://github.com/tiianhk/MaxMSP-MCP-Server

[16] https://github.com/nikmaniatis/Pd-MCP-Server

[17] https://docs.anthropic.com/

[18] https://github.com/ggerganov/llama.cpp

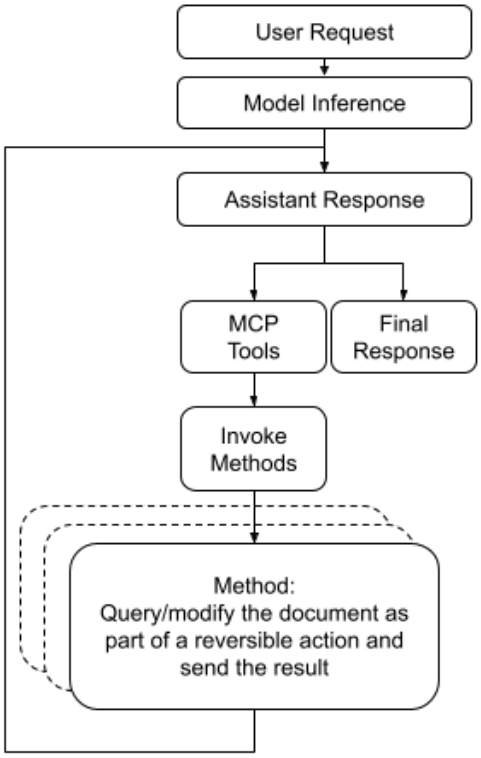


Figure 1. Flowchart of a request processing workflow using Neuralyzer, including model inference, MCP usage, reversible action management, and response feedback.

This mechanism allows the model to iteratively adjust its strategy until the desired result is achieved. The integration of Neuralyzer also relies on Partiels' native undo/redo mechanism. Any modification made via MCP is encapsulated in a reversible command. Thus, even in the event of an inadequate strategy or misinterpretation by the model, the user retains the ability to revert to a previous state of the document. Neuralyzer features a robust validation system for MCP tool calls built on top of llama.cpp, enabling precise identification of call errors (non-existent identifier, value outside the range, etc.). This enables the model to efficiently correct its calls without requiring user interaction.

3. NEURALYZER RESULTS AND FUTURE WORK

In February 2025, the first beta versions of Partiels integrating Neuralyzer were published on GitHub. [19] This version enables local inference of models compatible with MCP, as well as communication with Claude Desktop, as presented in the previous section. Here we present the ways in which Neuralyzer is used, the initial results observed, and future development plans.

3.1 Neuralyzer Usage Results

At this stage, no robust empirical evaluation methodology has yet been formalized. Our current focus is on the

[19] https://github.com/orgs/Ircam-Partiels/discussions/348

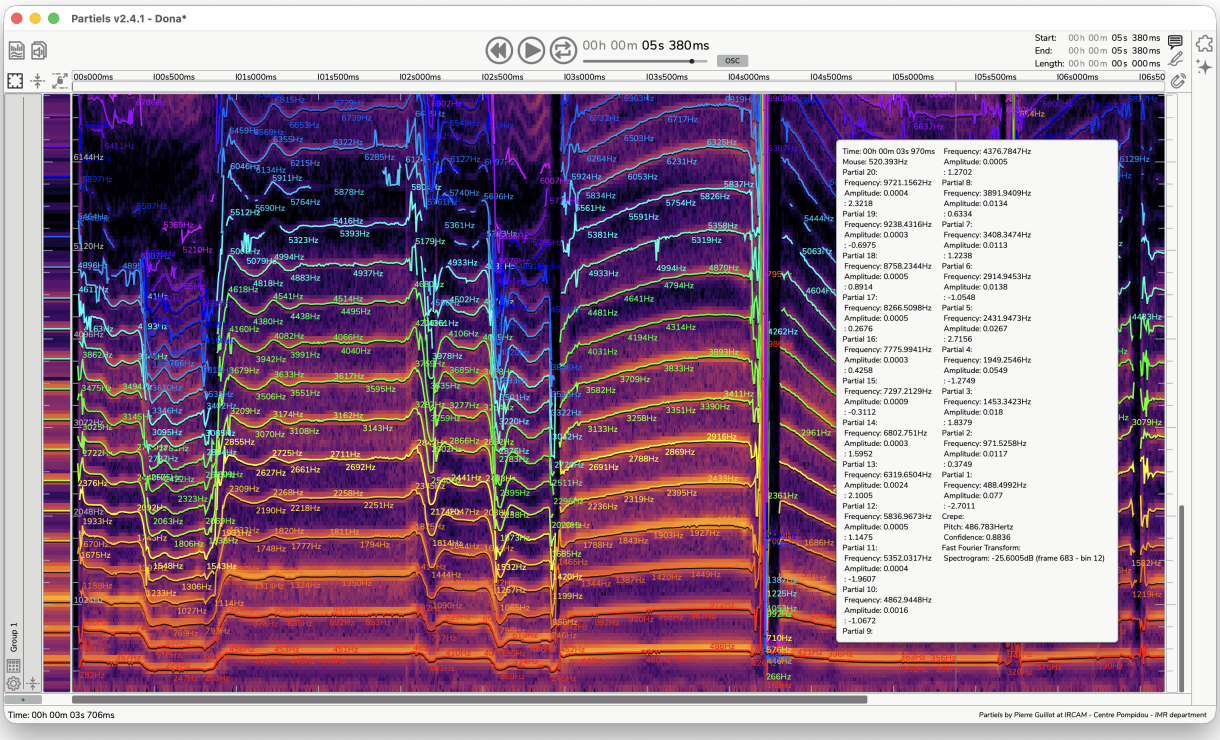


Figure 2. Overview of the Partiels application with a spectrogram, a fundamental-frequency analysis, and analyses of the first twenty harmonic partials produced through LLM inference and MCP generated with the prompt "*Perform the analysis of twenty partial harmonics*".

system's ability to respond appropriately to a given request within a reasonable time frame [20] while requiring little or no user intervention. [21] However, using Neuralyzer as an MCP server with Claude Desktop application provides an ideal benchmark for comparison, given the highly satisfactory results produced by this approach (moreover, this allows us to disregard the MCP engine, which is common to both approaches).

Tests focus in particular on two requests directly related to the issues presented in the first part:

1. Improve frequency precision while preserving the temporal precision of the spectrogram

2. Perform the analysis of twenty harmonic partials

In local inference, results depend strongly on model choice, [22] number of parameters, and context window size. Tests were notably carried out on a MacBook Pro M4 Max with 36 GB of RAM and the Qwen3.5 model [21] (Qwen3.5-35B-A3B-Q4_K_M.gguf [23]), which provides performance clearly superior to what was observed with Qwen2.5 or Qwen3 or Ministral 3, and the context size was 131,072 tokens, while the batch size was 2,048 tokens. Although results vary depending on the request, with this configuration, we achieve satisfactory results in almost all cases (with fewer than one in twenty attempts failing and fewer than one in ten attempts requiring a user modification).

The model performs very well for simple tasks such as creating a track for a specific analysis (such as pitch, transcription, or synchronization), adjusting graphical and analysis settings, and interpreting the results and the document's status. For instance, the first query results in a consistent increase in the window size and a corresponding adjustment of the overlap factor in order to maintain temporal resolution.

> *Assistant: The frequency precision of the spectrogram has been improved while preserving temporal precision by adjusting the parameters of the "Fast Fourier Transform" track. Specifically, the window size was increased to 50.0 ms, and the window overlapping was set to 4x. These changes enhance the frequency resolution without significantly compromising the temporal resolution.*

MCP tool calls may occasionally be malformed or inappropriate, however, Neuralyzer's response mechanism enables automatic correction and re-submission of the

[20] Ideally, a request should be processed in a matter of seconds or minutes, even when dealing with very large amounts of data.

[21] However, the user may be asked to clarify a request or confirm an action if the model is uncertain.

[22] Only certain recent models currently support the MCP protocol, but it is becoming a standard.

[23] `https://huggingface.co/unsloth/Qwen3.5-35B-A3B-GGUF`

request. Moreover, the model is more prone to errors when the request requires a sequence of dependent operations. In this case, it may be necessary to split the request into several steps. For example, the second request requires more intermediate steps: creating a track to estimate the fundamental frequency, creating multiple partial tracks using the PM2 plug-in, adjusting the "partialid" parameter for each track, selecting the appropriate input, and renaming the tracks (see Fig. 2). The model generally succeeds in developing a relevant strategy, although it sometimes requires additional input from the user, particularly in cases of errors related to plugin selection or document state synchronization. Despite these limitations, the results appear encouraging and justify continuing experiments in this direction. Although local models remain less effective than AI agents used by commercial services, primarily due to their more limited number of parameters, Neuralyzer compensates for these limitations through its model context management and inference strategy, particularly by providing the model with a precise description of the errors encountered. Moreover, the progress observed across different generations of models suggests a gradual improvement in robustness without a proportional increase in hardware resources, even though the requirement for a relatively powerful machine remains.

3.2 Neuralyzer Development Perspectives

Neuralyzer development perspectives are structured around four main axes: enrichment of MCP support, improvement of model parameterization, optimization of inference strategies, and evolution of deployment modalities.

3.2.1 Enrichment of MCP Support

A first direction involves enriching the data accessible through MCP. At this stage, the model mainly interacts with the document structure and analysis parameters. It has detailed access to the numerical results produced by the analysis engines, but this proves to be very resource-intensive and costly to use. It is necessary to better control the amount of data transmitted in order to avoid context overload. This may require structuring the results in a summarized or aggregated form that remains usable by the model.

In the same vein, integrating richer information about the plug-ins is a key lever. This information could be linked directly to the plug-ins via the IRCAM Vamp Extension (IVE) and made accessible to the model to improve the relevance of the proposed strategies.

Access to Partiels documentation and structured usage examples would also enhance the quality of the model's reasoning by reducing the amount of implicit inference.

3.2.2 Improvement of Model Parameterization

A second direction concerns improving model configuration for local inference. The current interface allows users to use custom models, beyond the default options, and to define the context and batch sizes, as well as the temperature and minimum probability parameters, which directly influence the stability, accuracy, and variability of the responses. Providing more comprehensive and advanced control over these parameters would allow Neuralyzer to be adapted to different use cases, whether educational, exploratory, or strictly technical. Similarly, offering the ability to choose quantization and the number of parameters in an exhaustive way could make Neuralyzer more accessible and optimize inference for a wider range of configurations.

3.2.3 Optimization of Inference Strategies

A third axis involves optimizing inference strategies. Fine-tuning approaches tailored to corpora related to digital audio analysis could improve understanding of specific concepts such as windowing, oversampling, or harmonic component analysis. Furthermore, the integration of Retrieval-Augmented Generation (RAG) mechanisms [22], based on the indexing of Partiels documentation and plugins, would reduce reliance on raw conversational memory and improve the relevance of responses while controlling the size of the transmitted context.

3.2.4 Evolution of Deployment Modalities

Finally, changes in deployment methods represent a key challenge. Local inference requires significant hardware resources, which can limit certain applications. Setting up a dedicated inference server, for example, at IRCAM, would allow for the pooling of resources and the provision of a centralized service. Such an architecture could also be adapted for educational settings, particularly to equip a classroom with a local server configured for multiple client workstations. This approach, however, raises questions regarding workload, maintenance, energy costs, and ethical considerations associated with the use of large-scale models.

4. CONCLUSION

Although still experimental, Neuralyzer already provides initial responses to the identified challenges and opens up new research directions. Initial experiments show that the use of local inference of language models can help reduce the complexity of learning to use a sound analysis tool and improve the user experience, particularly by limiting repetitive tasks, while adhering to ethical, environmental, and economic constraints. These results position Neuralyzer within an active research agenda at the intersection of software engineering, human-computer interaction, and AI applied to sound analysis.

5. ACKNOWLEDGMENTS

We thank Jean-Louis Giavitto, Serge Lemouton, and Miller Puckette for their constructive discussions on artificial intelligence, LLM usage, and software development in general. We also thank the users of the Partiels project for their technical support and valuable feedback during development and testing phases.

6. REFERENCES

[1] P. Guillot, "Partiels - exploring, analyzing and understanding sounds," in *Proceedings of the 28th International Conference on Digital Audio Effects (DAFx-25)*, Ancona, Italy, 2025.

[2] A. Vaswani *et al.*, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[3] T. B. Brown *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[4] C. Cannam *et al.*, "Sonic visualiser: An open source application for viewing, analysing, and annotating music audio files," in *Proceedings of the ACM Multimedia 2010 International Conference*, Florence, Italy, 2010.

[5] N. Bogaards and A. Robel, "An interface for analysis-driven sound processing," in *AES 119th Convention*, New York, USA, 2004.

[6] N. Bogaards *et al.*, "Sound analysis and processing with audiosculpt 2," in *Proceedings of the International Computer Music Conference*, Miami, USA, 2004.

[7] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023, pp. 28 492–28 518.

[8] G. Doras *et al.*, "A linear memory ctc-based algorithm for text-to-voice alignment of very long audio recordings," *Applied Sciences*, vol. 13, no. 3, p. 1854, 2023.

[9] M. Wright and A. Freed, "Open soundcontrol: A new protocol for communicating with sound synthesizers," in *Proceedings of the International Computer Music Conference*, Thessaloniki, Greece, 1997.

[10] M. Puckette, "Pure data: Another integrated computer music environment," in *Proceedings of the International Computer Music Conference*, Thessaloniki, Greece, 1997.

[11] A. Robel, "Estimation of partial parameters for non-stationary sinusoids," in *Proceedings of the International Computer Music Conference*, New Orleans, USA, 2006.

[12] OpenAI *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2024.

[13] A. Q. Jiang *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[14] A. Yang *et al.*, "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, 2024.

[15] H. Touvron *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[16] M. Abdin *et al.*, "Phi-3 technical report: A highly capable language model locally on your phone," *arXiv preprint arXiv:2404.14219*, 2024.

[17] Q. Team, "Qwen3.5: Accelerating productivity with native multimodal agents," February 2026. [Online]. Available: https://qwen.ai/blog?id=qwen3.5

[18] Y.-Y. Fan, "Mcp2osc: Parametric control by natural language," *arXiv preprint arXiv:2508.10414*, 2025.

[19] S. R. Group, "Audiofab: Building a general and intelligent audio factory through tool learning," *arXiv preprint arXiv:2512.24645*, 2025.

[20] S. Letz and Y. Orlarey, "Exposing FAUST applications to AI through MCP servers," in *Proceedings of the International Faust Conference (IFC-26)*. Cannes, France: International Faust Conference, Jun. 2026.

[21] J. Bai *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[22] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *arXiv preprint arXiv:2005.11401*, 2021.