AGENTIC MAX/MSP PROGRAMMING WITH LLMS AND MCP

Haokun Tian[1*] **Shuoyang Jasper Zheng**[1*] **Mathieu Lagrange**[3]
Stefan Lattner[2] **Charalampos Saitis**[1]
[1] Queen Mary University of London, UK [2] Sony Computer Science Laboratories, Paris, France
[3] Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, F-44000 Nantes, France
[*] equal contribution {haokun.tian, shuoyang.zheng}@qmul.ac.uk

ABSTRACT

The recently introduced Model Context Protocol (MCP) provides a standardised way for LLMs to interact with external tools. We implement an MCP server for Max/MSP, a visual programming language for music and multimedia, enabling LLMs to directly understand and generate Max patches, paving the way for flexible sound generation using LLMs. The LLM agent can dynamically retrieve official documentation of Max objects before taking actions, allowing for in-context learning. For understanding, it can explain user-selected objects. For generation, it can create objects, connect patch cords, send messages to objects, and more. This offers a more seamless way to assist Max users compared to traditional approaches where LLMs respond with text-based answers or code snippets. Moreover, it eliminates syntax errors caused by generating different formats of text-based code, as the process reduces to selecting the correct functions to call and providing the appropriate arguments. Code and demonstration are available online in a repository. [1]

1. INTRODUCTION

Max/MSP is a graphical programming language that targets the development of creative media systems (hereafter referred to as Max). This makes digital media programming more accessible to those who prefer non-text-based coding. Meanwhile, large language models (LLMs) have shown their ability in understanding and generating code. However, their application to Max is still a challenge. This is due to graphical programming languages like Max being structurally different from text-based code, and Max data is likely underrepresented in LLM training. Recently, Zhang et al. [1] evaluated how well LLMs generate Max patches in various forms, including raw Max file (`.maxpat`) in JSON and meta-programming approaches that produce code which, in turn, generates Max patches. However, these methods still have limitations, such as producing syntax errors and lacking integration with Max programming workflows.

Recently, Anthropic [2] introduced Model Context Protocol (MCP), an open standard that enables LLMs to connect and make use of external tools. [3] It is a communication layer between LLMs and tools, wrapping LLMs as clients and tools as servers. LLM applications such as Claude Desktop have native MCP client implementations. In this work, we implemented an MCP server for Max, allowing LLMs to retrieve information and program in the Max interface directly. Because the pipeline automates the process from prompts in LLM applications to actions in Max, it provides Max users with more seamless assistance. Moreover, this eliminates the need for models to adhere to the basic syntax of JSON or other meta-programming languages, as LLMs can perform desired actions as long as they select the correct functions and call them with the appropriate arguments.

2. AGENTIC MAX/MSP PROGRAMMING

Unlike approaches that generate Max patches in text-based code, MCP allows LLM agents to perform human actions in the Max programming interface. Actions from the agent, such as adding/removing objects and setting patch cords, are defined as tool functions in an MCP server. Below, we introduce all tool functions and index them sequentially in Section 2.1, describe the overall pipeline in Section 2.2, and demonstrate its usage in Section 2.3.

2.1 Features

In-context learning. This typically refers to giving an LLM a few example demonstrations in the prompt to guide it before asking it to perform the actual task [2]. Recent work extends this notion, showing that for coding tasks, LLMs can learn to use new libraries solely from generic descriptions (such as docstrings), without requiring carefully crafted task-specific demonstrations [3]. Additionally, DocPrompting shows that retrieving relevant documentation at inference time to guide code generation consistently improves performance [4]. Building on this idea,

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

[2] https://www.anthropic.com

[3] https://modelcontextprotocol.io

Late-Breaking / Demo Session Extended Abstract, ISMIR 2025 Conference

we make the official documentation of all Max objects accessible to the agent, providing it with in-context information about Max. For each object, the documentation contains its name, a natural language description, inlet and outlet types, and details about its arguments, methods, and attributes. Agents interact with the documentation through two tool functions: (i) returns a list of all valid Max object names, and (ii) retrieves the documentation for a single object. These allow the agent to dynamically retrieve and understand Max objects, enabling it to reason about object usage and verify its own actions with less hallucination.

Understanding. Within a Max patch, the agent can (iii) retrieve information about all existing objects or (iv) only those selected by the user. Accessing all objects helps the agent understand the overall structure and functionality of the patch, while focusing on the selected objects allows it to attend to what the user is interested in. The agent can (v) access every object's attributes and their corresponding values.

Generation. The agent can edit the Max patch directly. It can take actions including (vi-vii) adding and removing objects, (viii-ix) connecting and disconnecting patch cords, (x) setting object attributes, (xi) setting texts in message boxes, (xii) sending bangs to objects, (xiii) sending messages to objects, and (xiv) setting values in number or floating number objects. The agent decides whether to use each function and in what order to use them.

2.2 Pipeline

To bridge the LLM with Max, our implementation includes a two-stage communication system, as shown in Figure 1. First, the LLM application, which includes an MCP client, communicates with our MCP server via the Model Context Protocol. Second, the MCP server connects to Max through Socket.IO, [4] with a server running in Max using the Node for Max API [5] and a client in our MCP server. The MCP server is implemented in Python. Content manipulation and retrieval within Max patches is handled via Max's scripting features, while Max documentation is retrieved directly by the MCP server.

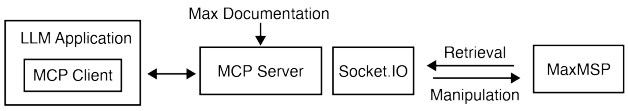

Figure 1. Our pipeline implementation.

2.3 Demonstration

The aim of our demonstration is only to show the technical availability of the communication pipeline, and we do not claim any benchmark. We refer readers to [1] for a benchmark that involves human evaluators. We tried an understanding task that involves a Sonnet 3.7 explaining a patch that implements ring modulation, including a full explanation of the patch as well as detailed descriptions of two selected objects within it. We also attempted a generation task in which we prompted Sonnet 3.7 to create a simple FM synth with an ADSR interface. We recorded two successful attempts, and links to the recordings can be found in our GitHub repository. However, trials and errors can occur in both tasks. For instance, in one of the unsuccessful generation attempts, the LLM miscounted the number of inlets in a `selector~` object. We encourage readers to test out the prototype based on their curiosity.

3. FUTURE DIRECTIONS

Fine-tuning. One research question is how to improve the ability of an LLM agent to generate Max patches through fine-tuning. We hypothesise that LLMs exhibit an imbalance between understanding and generation: they can reliably describe Max patches in natural language, but struggle with the inverse task of producing valid patches from textual descriptions. Following ToolLLM [5], we can leverage this asymmetry by constructing prompt–patch training pairs. Specifically, we use an LLM to generate descriptive prompts for Max patches, and then fine-tune a model to map prompts back to patches. For evaluation, we also adopt the LLM-as-a-judge approach, using an LLM to assess the correctness and quality of the generated patches.

Generalised sound matching. Sound matching, also known as the synthesiser inverse problem, aims to find a set of parameters for a synthesiser to reproduce a given target sound. Models can be trained either by directly regressing the target parameters [6] or by tweaking parameters to approximate the target audio via differentiable digital signal processing [7]. To our knowledge, existing methods assume a fixed synthesiser with a predetermined parameter set. In contrast, our agent-based programming approach exposes a more general formulation of the sound matching task: the agent must first generate a synthesiser, then parameterise it, and finally use it to render audio that matches the target sound. Crucially, the parameters are generated rather than regressed, which addresses the parameter symmetry problem, wherein multiple distinct parameter configurations can produce the same audio sample [8]. Based on this, a potential research direction is to train agentic LLMs to perform this generalised sound matching task.

Limitations. Max currently does not run on Linux, which prevents its use in training scenarios that require on-the-fly execution, such as online reinforcement learning. In such cases, Pure Data can serve as an alternative, which is open-source and runs on Linux. Moreover, an MCP server for Pure Data exists. [6] Additionally, more specialised synthesizers that can run on Linux like Vital [7] can be considered. Such a synthesiser is less general but could be easier to model due to a smaller parameter space.

4. ACKNOWLEDGEMENT

Haokun Tian and Shuoyang Zheng are supported by the EPSRC UKRI Centre for Doctoral Training in Artificial

[4] `https://socket.io`
[5] `https://docs.cycling74.com/apiref/nodeformax`
[6] `https://github.com/nikmaniatis/Pd-MCP-Server`
[7] `https://vital.audio`

Intelligence and Music (grant number EP/S022694/1).

5. REFERENCES

[1] W. Zhang, M. Leon, R. Xu, A. Cardenas, A. Wissink, H. Martin, M. Srikanth, K. Dorogi, C. Valadez, P. Perez *et al.*, "Benchmarking LLM code generation for audio programming with visual dataflow languages," *arXiv preprint arXiv:2409.00856*, 2024.

[2] Q. Dong, L. Li, D. Dai, C. Zheng, J. Ma, R. Li, H. Xia, J. Xu, Z. Wu, B. Chang, X. Sun, L. Li, and Z. Sui, "A survey on in-context learning," in *Empirical Methods in Natural Language Processing*, 2024.

[3] A. Patel, S. Reddy, D. Bahdanau, and P. Dasigi, "Evaluating in-context learning of libraries for code generation," in *North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2024.

[4] S. Zhou, U. Alon, F. F. Xu, Z. Wang, Z. Jiang, and G. Neubig, "DocPrompting: Generating code by retrieving the docs," in *International Conference on Learning Representations*, 2023.

[5] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian *et al.*, "ToolLLM: Facilitating large language models to master 16000+ real-world apis," in *International Conference on Learning Representations*, 2024.

[6] F. Bruford, F. Blang, and S. Nercessian, "Synthesizer sound matching using audio spectrogram transformers," in *International Conference on Digital Audio Effects*, 2024.

[7] J. Engel, L. H. Hantrakul, C. Gu, and A. Roberts, "DDSP: Differentiable digital signal processing," in *International Conference on Learning Representations*, 2020.

[8] B. Hayes, C. Saitis, and G. Fazekas, "Audio synthesizer inversion in symmetric parameter spaces with approximately equivariant flow matching," in *International Conference on Music Information Retrieval*, 2025.