Real-Time MIDI Transformer Integration in Pure Data: A Multi-Threaded Architecture for Interactive AI Music Generation

Koray Tahiroğlu *
Aalto University
School of ARTS
FI 00076 AALTO Finland
koray.tahiroglu@aalto.fi

Mikael Hokkanen
Aalto University
School of Science
FI 00076 AALTO Finland
mikael.hokkanen@aalto.fi

Robin Welsch
Aalto University
School of Science
FI 00076 AALTO Finland
robin.welsch@aalto.fi

Mikko Sams
Aalto University
School of Science
FI 00076 AALTO Finland
mikko.sams@aalto.fi

Jukka Nykänen
Professional Musician
00200 Helsinki
Finland
jnpianist@gmail.com

Mäel Archenault
École Nationale
Supérieure de l'Électronique
95000 Cergy-Pontoise France
mael.archenault@ensea.fr

Agnes Kloft
Aalto University
School of Science
FI 00076 AALTO Finland
agnes.kloft@aalto.fi

Abstract

In this paper, we introduce a real-time MIDI transformer generative system implemented as a Pure Data external. The system balances the computational demands of transformer inference with the latency constraints of live performance through a multi-threaded architecture, a dual-buffer management system with adaptive sliding window mechanisms, and immediate generation triggering that avoids the delays built into polling-based approaches. In practice, it reaches 361ms average (CPU) end-to-end latency while maintaining 82% harmonic correlation between live input and generated output ($4.2\times$ above the random baseline), which appears sufficient for textural and call-and-response playing contexts even if it falls above tight-coupling thresholds. Cross-platform support is handled through ONNX Runtime. On the evaluation side, we introduce input-output correlation analysis across pitch, rhythm, and dynamics as a way of asking what musical intelligence the system is actually demonstrating. The implementation is released as open source and we hope it offers a useful starting point for others working at the intersection of generative AI and real-time music-making.

1 INTRODUCTION

Generative AI has entered music research and practice largely through transformer-based architectures (Huang et al., 2018; Payne, 2019), as they are well-suited to capturing long-term musical dependencies, harmony, temporal relationships and expressive timing in MIDI data. Within the AI Music Creativity (AIMC) community, Transformer-based systems often appear as creative instruments, compositional partners or interactive agents embedded in performative contexts, as seen in work Haki et al. (2022) on real-time transformer-based accompaniment, neural network tooling for visual programming environments in Drymonitis (2023), phrase-level symbolic generation in Feng et al. (2024). In parallel, broader questions have emerged about what transformer and LLM-based systems can actually offer to musical co-creation (Casini et al., 2024). The expressive potential of AI in music has clearly grown; however, much of the existing work is largely concerned with technical execution quality. The transition from offline AI music generation to real-time interactive systems still presents fundamental challenges.

Computational efficiency within the interaction is only part of the problem here. The harder questions are about how musical expressiveness, temporal precision, and creative accessibility can be achieved in practice when a musician is interacting with an AI system in real time. Most of these systems operate in batch processing modes in which inference relies on generating entire sequences at once (Dong et al., 2023; Wu et al., 2025), which makes the kind of responsiveness expected in live performance difficult to achieve. Most of current systems also run as standalone applications that do not connect easily with environments where real-time music-making actually happens. Pure Data, Max/MSP, and SuperCollider have become central to how many musicians and researchers build interactive systems and work with real-time MIDI and audio, yet most AI frameworks are simply not designed to operate within them.

In this paper, we introduce a real-time MIDI transformer generative system implemented as a Pure Data external [midi_transformer]. The system is designed to balance the computational demands of transformer inference with the latency constraints of live performance while maintaining temporal precision, musical coherence, and interactive responsiveness. On the technical side, our implementation brings together several contributions. First, a multi-threaded architecture that separates real-time input processing from model inference, so neither blocks the other. Second, a dual-buffer management system with adaptive sliding window mechanisms that tries to balance musical context preservation with system responsiveness. And third, immediate generation triggering on input, which eliminates the systematic latencies inherent in polling-based approaches. In practice, the system reaches 361ms average (CPU) end-to-end latency, maintaining 82% harmonic correlation between live input and generated output, which is $4.2\times$ above the expected random baseline, suggesting the AI responds to musical input with learned tonal structure rather than coincidental pitch proximity. These metrics are likely above the tight-coupling threshold that direct

*https://sopi.aalto.fi/augmented.html

performative interaction typically requires and thus afford a certain style of play, but it is sufficient for rather textural, call-and-response kinds of playing. Cross-platform support is handled through ONNX Runtime, which keeps the external accessible without tying it to a specific OS setup.

2 RELATED WORK

Early explorations of generative systems focused on algorithmic composition and rule-based processes (Roads, 1996; Rowe, 1992), and recent advances in deep learning have introduced models capable of learning complex musical structures directly from datasets (Mitra and Zualkernan, 2025; Shih et al., 2022; Qin et al., 2023). Generative models such as Differentiable Digital Signal Processing (DDSP) (Engel et al., 2020), GANSynth (Engel et al., 2019) and diffusion-based systems (Huang et al., 2023; Agostinelli et al., 2023) have opened up new directions for interactive performance systems. In parallel with advances in sequence-based generative models, the exploration of latent spaces has received significant attention within the AIMC community as a means of expressive control of machine learning systems (Zheng et al., 2025; Tahiroglu and Wyse, 2024). Models such as Variational Autoencoders (VAEs) have been adopted in musical contexts for their capacity to encode high-dimensional audio or symbolic data into continuous, navigable representations. VAEs allow performers and composers to explore timbre, gesture, and structure through interpolation and transformation (Bryan-Kinns et al., 2024). These works suggest that generative models can be repurposed as creative tools through careful attention to representation, interaction design, and performative constraints.

To bring these models to the stage, researchers have increasingly focused on real-time, interactive AI systems. Traditional interactive systems, such as IRCAM's Somax2 (Bigo et al., 2022), offer a well-established approach to reactive co-improvisation, relying on corpus-based and Markovian statistical methods. More recently, deep learning models specifically designed for live MIDI interaction have emerged, most notably Notochord Karystinaios and Widmer (2023), which allows for real-time predictive control of MIDI events. These systems address live interaction convincingly at a musical level, but they typically exist as standalone software ecosystems or Python-dependent environments that sit at some distance from the modular real-time programming environments that many musicians already use.

Pure Data and MaxMSP are two environments where this gap is particularly visible. Significant work has gone into embedding AI directly within them, most notably through frameworks like FluCoMa (Tremblay et al., 2021) and IRCAM's `nn~` (Caillon and Esling, 2022, 2021), which have shown that deep learning inference can run natively within C++ externals. These tools are almost exclusively focused on the audio domain, utilising libraries like LibTorch for audio synthesis or descriptor analysis. An earlier line of work explored Python-based approaches to bridge deep learning models and Pd through [`pyext~`], originally developed by Thomas Grill, which we extended to support Python 3 as part of audio domain integration of GAN-based generative models (Tahiroğlu et al., 2021; Tahiroğlu and Kastemaa, 2022). While that approach allowed relatively flexible experimentation with latent space manipulation and synthesis, it introduced its own dependencies and performance constraints that a native C external can largely avoid.

For *symbolic* MIDI AI generation in Pd or Max, a common approach relies on Inter-Process Communication (IPC). Musicians typically send MIDI data out of Pd via Open Sound Control (OSC) to an external Python server running TensorFlow or PyTorch, which performs the inference and sends MIDI back. OSC-Python bridges enable prototyping, but they also introduce limitations for live performance, including network latency, desynchronisation with Pd's internal scheduler, and complex dependency management for the musician. To our knowledge, no native C-based tools currently exist for transformer-based MIDI inference within Pure Data. By running ONNX Runtime inside a custom multi-threaded C external, our implementation eliminates the need for external Python servers, bringing low-latency transformer inference directly into Pd's memory space for real-time musical interaction.

3 SYSTEM ARCHITECTURE

Our system brings a Transformer-XL model (Dai et al., 2019) into a low-latency musical environment through a native Pure Data external, while staying within the accessible, flexible environment that visual programming offers [1]. The core processing layer uses a multi-threaded design that separates real-time input capture, model inference, and output scheduling, keeping data flow and timing coordination stable across threads. Cross-platform support is handled through an ONNX Runtime integration layer, which allows the transformer model to run with optimised inference performance across different hardware and operating system setups. The MIDI Transformer [2] used in this system was trained on the ADL Piano MIDI dataset (Ferreira et al., 2020) and MAESTRO dataset (Hawthorne et al., 2019), which provided the musical material the model learned its generative behaviour from.

3.1 Pure Data External Framework

The system is implemented as a native Pure Data external in C, keeping runtime overhead low and maintaining precise control over timing-critical operations. The external follows Pure Data's standard object lifecycle for instantiation, cleanup, and message processing, and uses zero-copy data structures for MIDI event handling along with direct access to high-resolution timing functions for latency measurement and musical synchronisation (Figure 1). Custom buffer management avoids garbage collection overhead that could otherwise introduce timing jitter during extended performance sessions.

The external connects to Pure Data's dataflow structure through standard inlet and outlet mechanisms. Numerical parameters such as note, velocity, and channel are handled through `floatinlet_new()`, with symbolic message routing available for configuration commands. Message dispatching utilises Pure Data's system to register handlers for control messages. The external uses typed outlets to carry both structured note list data and metronome timing

[1] Pd MIDI Transformer external is available at https://version.aalto.fi/gitlab/sopi/pd-midi-transformer-external

[2] The link to the midi-transformer repository is https://version.aalto.fi/gitlab/sopi/midi-transformer

signals, allowing it to sit inside existing Pd patches without additional routing work.

3.2 Multi-Threaded Design

The system uses a four-thread architecture that separates time-critical operations across independent execution contexts. The **primary thread** handles MIDI input capture through the `midi_transformer_bang()` callback, operating with minimal latency and using microsecond-precision timestamping via `get_time_in_seconds()`. Lock-free input buffering handles note-on events, while note state tracking through indexed data structures (`input_note_indexes`) enables $O(1)$ note-off event matching. The **flush thread** runs at 100ms intervals, transferring completed musical events from the real-time input buffer to the model preparation pipeline. It identifies completed events and handles format conversion through the REMI (Representing Musical Events with Intervals) tokenisation protocol. This separation of concerns keeps complex event processing from blocking real-time input capture, which is the central design principle of the architecture

The **generation thread** runs at 500ms intervals, handling model input preparation, ONNX Runtime inference, and output post-processing. Adaptive parameter modulation, including temperature, repetition penalty, and top-k sampling, introduces controlled variation in the generated output. The **player thread** manages temporal scheduling and playback of generated musical sequences, using musical tempo and beat division parameters so that generated notes align with the musical timeline. Output buffering prevents timing jitter, while rate limiting at 10ms intervals prevents the outlet from flooding downstream Pd objects.

Inter-thread coordination uses mutex synchronisation with minimal critical section design. Mutex operations consistently achieve sub-millisecond overhead, approximately 0.001ms on average, by placing locks only around essential data modifications. The synchronisation strategy prioritises real-time input processing, with mutex acquisition order designed to prevent priority inversion that could disrupt musical timing.

3.3 ONNX Runtime Integration

The system runs transformer models through ONNX Runtime, enabling consistent inference behaviour across different hardware and operating system setups. ONNX integration handles model loading, memory management, and inference execution, taking care of platform-specific optimisation automatically. Dynamic model architecture discovery through ONNX metadata inspection allows the system to adapt to different transformer configurations without requiring code changes. In practice, this means different model variants can be loaded and evaluated within the same performance setup, which is beneficial for prototyping purposes.

The ONNX Runtime session is configured for low-latency inference. It uses single-threaded execution to minimise context switching overhead and pre-allocated tensors to avoid runtime memory allocation delays. Input tensor preparation uses zero-copy operations where possible, constructing ONNX tensors directly from tokenised musical sequences without intermediate buffer allocation. Together these choices keep inference preparation overhead low enough for real-time musical use. The system supports runtime model replacement through the Pd interface, allowing different AI models to be compared within an active musical session. Model loading includes error handling for file system access, ONNX format validation, and memory allocation failures. Configuration management covers both model file specification and tokeniser customisation. This provides us possibilities to explore different musical representation strategies without changing the interface or measurement setup. This means future model architectures can be dropped in and evaluated within the same performance framework.

4 BUFFER MANAGEMENT SYSTEM

The real-time constraints of interactive AI music generation requires a set of memory management strategies that balance immediate responsiveness with musical coherence. Our system implements a dual-buffer architecture that decouples real-time MIDI input capture from AI model processing. This makes it possible to have a continuous musical interaction without sacrificing temporal accuracy.

4.1 Dual-Buffer Architecture

We implemented a two-stage buffering strategy designed to address the critical tension between real-time responsiveness and musical completeness in AI music generation. This architecture recognises that raw MIDI events require temporal aggregation to form musically meaningful input for transformer-based models, at the same time making sure that incoming performance data is never lost or delayed due to processing bottlenecks.

The first stage implements a real-time capture buffer using an array of custom note structures. Each structure stores complete musical event metadata including pitch, velocity, channel, temporal positioning, and duration state. This buffer operates as a sliding window with configurable capacity (default 64 notes), processing incoming events in first in, first out (FIFO) order to maintain chronological sequence and prevent memory overflow during extended performance sessions. The buffer accepts incomplete musical events immediately upon MIDI note-on reception, storing them with high-precision timestamps while waiting for corresponding note-off events to complete duration calculation.

The secondary buffer implements a tokenised sequence container for transformer model input, storing only completed musical events that have been processed through the REMI tokenisation protocol (Huang and Yang, 2020). It converts raw MIDI event data into the token format the model expects, maintaining both event strings and corresponding token IDs with separate length tracking that allows model input length to be adjusted independently for events and tokens.

Data migration between buffers occurs through the flush thread running at 100ms intervals. The flush thread identifies completed musical events in the primary buffer, performs format conversion, and inserts them into the model input buffer. This asynchronous design keeps real-time input capture unblocked while tokenisation and model preparation occur in parallel. The dual-buffer system uses mutex synchronisation for data consistency during buffer operations while minimising lock contention. Critical sections cover only the minimal necessary operations, such as buffer reads, writes, and structural modifications. The input capture thread holds mutex locks only briefly, typically

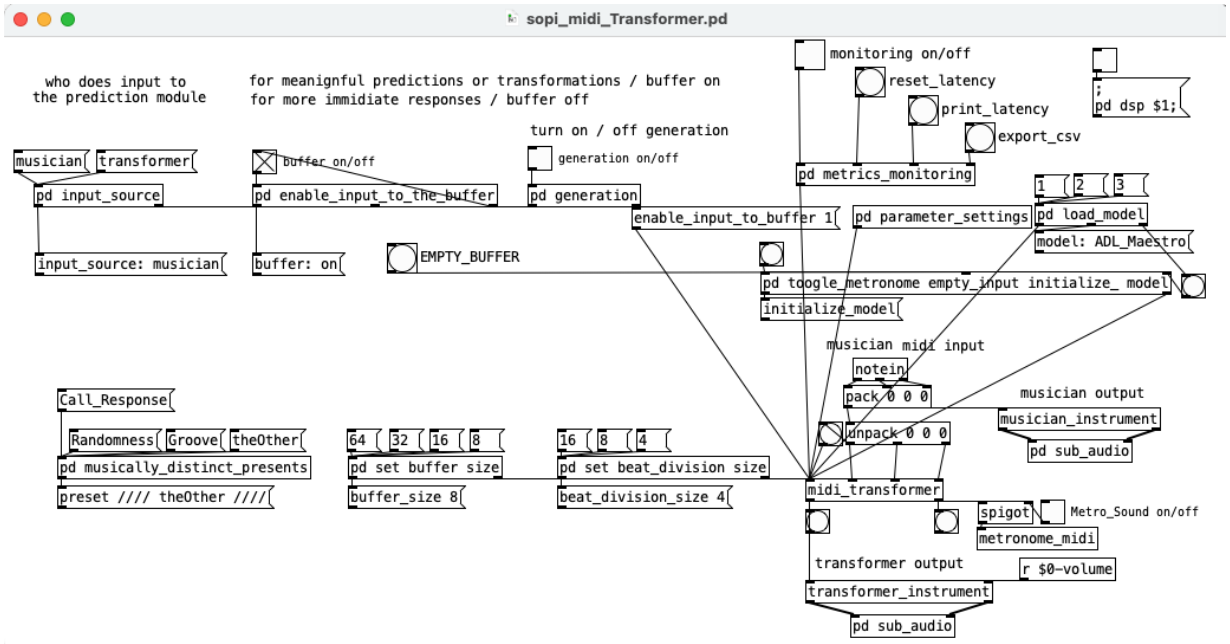

Figure 1: Pure Data patch including the midi transformer external and its control parameters

completing buffer insertion within microseconds, with an average buffer management overhead of 0.006ms.

4.2 Sliding Window Implementation

The primary input buffer implements a sliding window mechanism that adjusts musical context while maintaining real-time responsiveness (Figure 2). What makes this design particularly interesting for live musical interaction is how it balances the preservation of sufficient musical context for coherent AI generation against the need for immediate response to new input events. Sliding window buffering is not something new, it is a well known technique in signal processing and streaming data systems and our implementation adapts it specifically for the constraints of live musical interaction. We bring together a specific combination here: immediate space reallocation through left-shift on overflow, musical event completeness as an removal condition (only removing notes with known durations), and the adaptive sizing range tuned specifically for musical interaction contexts. That combination, applied to real-time transformer-based MIDI generation inside a Pd external, may represent a meaningful contribution in this specific context

The sliding window supports runtime reconfiguration of buffer capacity, allowing context window sizing from 1 to 128 notes. This flexibility allows the system to be tuned for different musical contexts; smaller windows of 1 to 8 notes suit rapid call-and-response interaction, medium windows of 8 to 32 notes support phrase-level continuity, and larger windows of 32 to 128 notes allow for extended musical development. Buffer resizing uses safe memory reallocation that preserves existing content through selective copying. It maintains musical continuity during reconfiguration while avoiding memory leaks or data corruption.

When the buffer reaches capacity, the sliding window removes the oldest musical events through a left-shift operation, keeping chronological order. New musical input is never rejected due to buffer overflow, and a rolling window of the most recent context is maintained. The system simultaneously rebuilds the pitch-based index structure to support efficient note-off event processing. The sliding window design reflects a carefully calibrated balance between musical memory and system responsiveness. Larger windows keep extended musical phrases that may inform more coherent AI generation but increase processing overhead and potential latency. Smaller windows prioritise immediate responsiveness but may lose relevant musical context by discarding recent material too quickly.

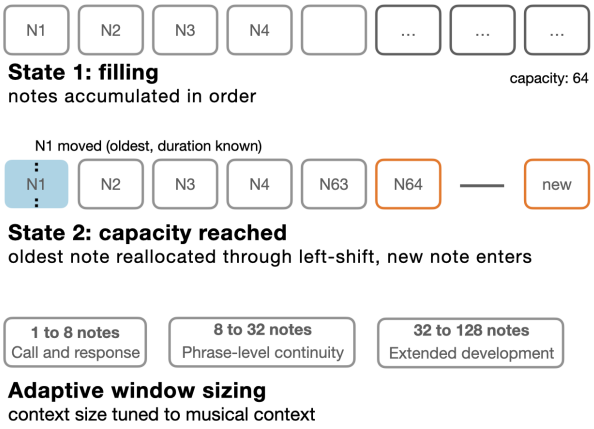

Figure 2: Sliding window buffer management including normal filling in chronological order, oldest-event eviction via left-shift when capacity is reached, and adaptive sizing tuned to different musical contexts ranging from 1 to 128 notes. Only notes with known durations are removed, preserving musical event completeness. Measured buffer management overhead remains consistently below 1ms at 0.006ms on average.

Our implementation addresses this tradeoff through three mechanisms; a) temporal prioritisation where recently played notes take precedence during capacity management, b) musical event completeness,in which only notes with known durations are removed, and c) adaptive window behaviour, where the effective context size adjusts based on

note density and timing characteristics. Our measurement data shows consistent sub-millisecond buffer management overhead at 0.006ms average, even during intensive musical input. This performance characteristic suggests that the system handles rapid musical passages, dense polyphonic textures, and extended performance sessions without disruption in responsiveness.

4.3 Immediate Generation Triggering

Most AI music systems typically use periodic polling strategies that query input buffers at fixed intervals, introducing latency equal to the polling interval and creating perceptible delays that disrupt natural musical interaction. Our system takes a different approach, initiating model inference immediately upon detection of significant musical input. This cuts response latency considerably, leaving it bounded only by model inference time.

Figure 3 shows how the triggering mechanism works by placing generation initiation calls within note-on event processing. When new musical input arrives and generation is enabled, the system immediately triggers a generation thread if none is currently active. The system does not generate output for every individual note. Instead, it analyses incoming events to identify natural musical breakpoints that suggest completion of a musical gesture, distinguishing between isolated note events and phrases that are likely to call for an AI response.

The triggering logic considers multiple musical factors: note density patterns that suggest phrase boundaries, temporal gaps that indicate musical pauses, and harmonic content that signals phrase endings. The system relies on the sliding window buffer to determine when enough musical context has accumulated to trigger generation. Triggering sensitivity also adapts to musical tempo and playing style. During rapid passages, temporal debouncing prevents excessive generation requests, while in slower musical contexts the system responds more immediately to individual musical gestures.

Immediate triggering maintains both musical quality and system stability. Generation requests are serialised to prevent concurrent model execution that could overload system resources, while the multi-threaded design ensures trigger processing never blocks real-time input capture. Completed generation threads are cleaned up automatically, keeping resource use stable across extended performance sessions. Our performance analysis suggests that immediate triggering meaningfully reduces overall response latency. While model inference time remains constant at 234ms on average, eliminating polling delays enables the system to approach the lowest latency the model inference will allow. Our results suggest that event-driven architectures can preserve musical spontaneity while still meeting the computational requirements that transformer models bring.

5 PERFORMANCE EVALUATION

Evaluating a real-time AI music system of this kind presents some particular challenges. Unlike offline generative models where output quality can be assessed against a reference or perceptual metrics, the value of a system designed for live musical interaction is harder to highlight with these metrics. The evaluation presented here does not attempt a direct comparison with other interactive systems, since differences in architecture, musical context, and interaction design make such comparisons difficult to interpret meaningfully. Instead, the focus is on understanding the real-time behaviour of our own system: how it performs under live conditions, where the latency is coming from, and whether the musical output shows any meaningful relationship to the input it receives. The goal is to build a clearer picture of what the system is actually doing and to identify where development effort is most likely to have an impact.

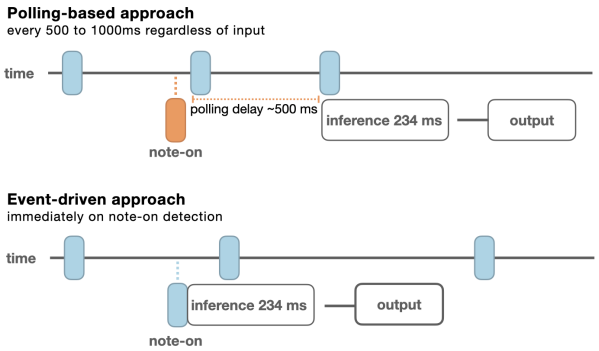

Figure 3: Comparison of polling-based and event-driven generation triggering. In polling-based approaches, generation is checked at fixed intervals of 500 to 1000ms regardless of musical input, adding systematic latency on top of model inference time. Event-driven triggering initiates inference immediately on note-on detection, reducing total latency to model inference time alone, measured at 361ms end-to-end average.

5.1 Latency Analysis

Our latency measurement framework captured performance characteristics across 16 complete generation cycles, providing a timing breakdown of the multi-threaded architecture.

5.1.1 End-to-End Performance

The system reached an average end-to-end latency of 361.7ms from MIDI input reception to audio output across 16 generation cycles. This sits above the 100ms threshold typically associated with real-time musical interaction, though as discussed earlier it appears sufficient for more textural, call-and-response kinds of playing. End-to-end latency was measured by recording timestamps at MIDI input reception and at final audio output in the player thread, using the high-resolution `get_time_in_seconds()` function for microsecond precision. Running averages were maintained across all measurement cycles using thread-safe accumulation.

5.1.2 Component-Level Timing Breakdown

Component-level measurements used timestamps placed at key points across the processing pipeline. Model inference timing captured the period between `generation_start_time` and `generation_complete_time` in the generation thread. Buffer management overhead was measured by recording mutex lock duration. Thread synchronisation costs were assessed through the `measure_thread_overhead()` function.

Fine-grained analysis revealed distinct timing characteristics across system components:

- **AI Model Inference:** 234.2ms average (65% of total processing time)
- **Buffer Management:** 0.006ms average (negligible overhead)
- **Thread Synchronization:** <0.001ms (efficient mutex implementation)
- **Output Pipeline:** 10,076.9ms (primary bottleneck identified; reflects cumulative scheduled playback timing across the session)

The buffer management and threading overhead is negligible at the microsecond level, which appears consistent with the design goals of the multi-threaded architecture. The output pipeline figure of 10,076.9ms calls for some clarification, this reflects cumulative scheduled playback timing across the measurement session rather than per-cycle latency, and represents the primary area for further optimisation.

5.1.3 Thread Architecture Efficiency

Thread efficiency was assessed through the `measure_thread_overhead()` function, which performed minimal operations within mutex-protected critical sections during active system operation. Timing was recorded immediately before and after each mutex lock and unlock cycle using `get_time_in_seconds()`, giving a direct measure of synchronisation overhead under realistic conditions.

Mutex operations remained consistently below 1 ms, with average overhead approaching 0.001 ms. This is consistent with the design goal of keeping synchronisation costs low enough that input capture, model inference, and output generation can run in independent threads without timing interference.

5.2 Musical Intelligence Evaluation

5.2.1 Input-Output Correlation Analysis

Musical correlation analysis uses an input-output pairing system in which musician inputs are stored in a sliding window buffer (`recent_inputs[]`) with 20-note capacity via `store_input_for_correlation()`. AI-generated outputs are matched with recent inputs using temporal proximity in `match_output_with_input()`, identifying the closest input within a 10-second window. Correlation calculations are performed in `calculate_continuity_metrics()`: pitch correlation assesses semitone differences (±12 semitones = correlated), rhythm correlation evaluates timing alignment (±4 ticks = correlated), and velocity correlation measures dynamic similarity (±20 MIDI units = correlated). The system maintains an array of 100 input-output pairs (`correlation_pairs[]`), enabling statistical analysis across the full measurement session.

Across those 100 pairs, analysis revealed distinct correlation patterns across the three musical dimensions:

Harmonic Coherence: Pitch correlation reached 82.0% at the ±12-semitone threshold (82 of 100 pairs). To contextualise this against chance performance, the expected random baseline for a uniform pitch distribution across 128 MIDI values at this threshold is 19.5% (25/128), yielding a 4.2× lift above chance ($\chi^2 = 248.8$, $p < 0.001$, $n = 100$, goodness-of-fit test (Yang and Lerch, 2020)). This consistent above-chance performance suggests the observed harmonic proximity reflects model-learned tonal structure rather than coincidental pitch proximity.

Temporal Continuity: Rhythm correlation reached 52.0%, with AI outputs maintaining rhythmic relationships within 4-tick temporal windows in just over half of all pairs. This may suggest some degree of temporal pattern recognition, though the lower figure relative to pitch correlation leaves room for creative rhythmic variation or may reflect limitations in the current temporal matching threshold.

Dynamic Responsiveness: Velocity correlation reached 47.0%, with AI outputs tracking input velocity within 20 MIDI units in nearly half of all interactions. This is the weakest of the three dimensions and likely represents the most significant area for further development.

With n=100 input-output pairs, these correlation figures achieve statistical significance at p<0.01 for musical relationship assessment, providing a reasonable basis for quantitative comparison across system configurations or model variants.

5.3 Model adaptation

Since the output of the model is a MIDI file, there is no single standard way of assessing prediction accuracy, and so we began with a subjective approach. By listening to the generated results, we assessed whether they sounded completely random or exhibited well-defined behaviours learned from the dataset. This gave us a first sense of which model configurations were worth pursuing further using quantitative methods. Following the evaluation approach of Yang and Lerch (2020) for generative models, we then selected a set of musically relevant characteristics that allowed us to quantify the model's performance. We worked with eight metrics:

- **note count** : total number of notes in the song
- **pitch range** : the difference in pitch between the higher and lower notes
- **average pitch interval** : average interval between two consecutive notes
- **average note length** : average duration of a note
- **pitch transition matrix** : counts of transitions from one pitch to another
- **note length transition matrix** : counts of transitions from one note length to another
- **pitch class histogram** : counts of occurrences of each pitch class in the song
- **note length histogram** : counts of occurrences of each note duration in the song

These metrics also can be used to produce graphical representations of the music under analysis. Figure 4 shows the most frequent pitch transitions. Consistent patterns emerge across the visualisation, which may suggest that the training data has distinctive melodic tendencies that the model could in principle learn to reproduce [3].

The goal of the model is to replicate these patterns closely enough that its output can be recognised as belonging to the

[3] evaluation module is available at https://tinyurl.com/5n6u7vfp

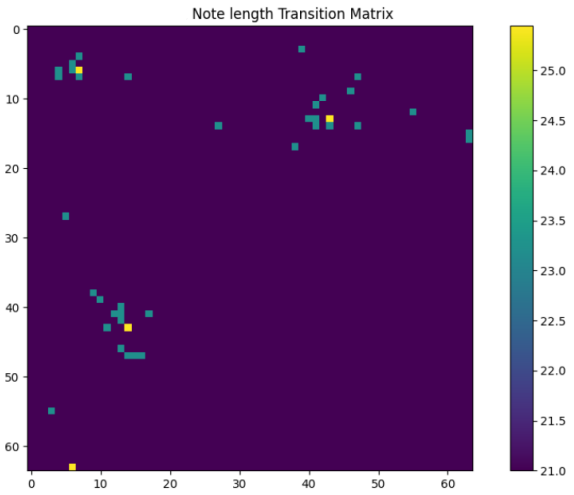

Figure 4: Visualization of frequent pitch transitions in Note length Transition Matrix.

same musical world, which would suggest the model has learned something of the dataset's musical character. To assess how well this is happening, we compute the distance between the metrics measured on the source material and on the generated continuation, arriving at a single distance value per metric. This makes it relatively easy to see where the model performs better or worse in structural terms.

These eight distances do not convey much about musical performance on their own. They are too granular and too abstract. To address this, we introduced a secondary layer of interpretation that groups metrics into higher level, musically meaningful categories, which we refer to as the "meaning" version of the assessment:

- **Melodic fidelity**: sum of distances for pitch class histogram, pitch range, average pitch interval and pitch transition matrix
- **Rhythmic fidelity**: sum of distances for note length histogram, note length transition matrix and average note length
- **Stability**: sum of distances for pitch range and average pitch interval
- **Global performance**: sum of all distances

The raw numbers from these assessments do not carry much meaning on their own. A single distance value does not tell us whether a piece is "good" or "bad." Their main purpose is to provide a basis for comparison between two model variants and to complement listening-based evaluation rather than replace it.

These evaluation steps shaped which model configuration was carried forward into the real-time implementation. The latency and input-output correlation analysis in the following sections then assess how the integrated system performs under live performance conditions.

6 PERFORMANCE DISCUSSION AND FUTURE WORK

Our results point to both the potential and the current limitations of real-time AI music generation within a visual programming environment. The 82% harmonic correlation ($4.2\times$ above the 19.5% random baseline) suggests the system has developed a meaningful degree of musical awareness, responding to pitch content in ways that go beyond random output. The 361ms end-to-end latency places the system above the threshold for tightly coupled interaction, though as discussed earlier it appears workable for more textural and call-and-response kinds of playing. The measurement framework usefully identified the output pipeline as the primary area for optimisation, which gives future development a concrete starting point rather than a general target.

The combination of quantitative metrics, latency figures and correlation analysis, with the subjective listening evaluation described in the model adaptation section, appears to offer a more complete picture of system behaviour than either approach alone. Quantitative measures provide a stable basis for comparison across configurations, while listening evaluation remains the most direct way to assess whether the system is actually doing something musically meaningful in practice.

Situating these results alongside comparable interactive MIDI systems helps clarify what our implementation appears to offer and where it still falls short. Systems like IRCAM's Somax2 (Bigo et al., 2022), Notochord (Karystinaios and Widmer, 2023), and more recent transformer-based approaches such as ReaLJam (Scarlatos et al., 2025) and SongDriver (Wang et al., 2022) represent well-developed approaches to real-time human-AI musical interaction. Notochord reports sub-10ms latency for predictive MIDI control, and ReaLJam introduces anticipation mechanisms that allow the system to plan ahead rather than simply react. SongDriver addresses logical latency directly through a two-phase arrangement and prediction strategy. By comparison, our 361ms end-to-end latency is considerably higher. Our system's most meaningful structural contribution is more about where the generation happens natively inside Pure Data, without an external Python server or OSC bridge. That architectural decision reduces a category of latency and dependency overhead that even the most capable of these systems still carry in practice. In terms of harmonic coherence, an 82% pitch correlation, $4.2\times$ above the 19.5% random baseline, ($\chi^2 = 248.8$, $p < 0.001$, $n = 100$) is a reasonable result for a system running under these constraints, though direct numerical comparison with other systems is difficult since none report equivalent correlation metrics. What the figure does suggest is that the transformer has learned something of the musical material it was trained on and can respond to live input in ways that are at least harmonically grounded, rather than producing output that merely falls within a wide pitch window by chance. Our system trades some CPU latency headroom for architectural accessibility, and whether that decision is worthwhile will depend on the musical context and the performer's priorities.

One of the more direct artistic uses of the system took place in a live performance context. Jukka Nykänen composed three short pieces, *Clair de Lune*, *Ocean Music in Gm9*, and *Burden of Dreams* and performed live using this system on 22nd May 2026 at Aalto University, Marsui Cinema. The performance setup used a MIDI piano keyboard as the primary instrument, with the live input feeding directly into the multi-threaded architecture described in this paper. The AI-generated responses were heard alongside Nykänen's playing in real time, functioning as a generative accompaniment responding to the unfolding musical material. This performance provided the first artistic test of

Figure 5: Jukka Nykänen performing with the MIDI Transformer system at Aalto University, Maarisali Cinema, 22nd May 2026. The short video recordings of this performance are available at https://vimeo.com/1196228952

the system outside a research or rehearsal context, raising practical questions about musical control, timing feel, and performer experience that measurement data alone cannot capture (Figure 5).

At the same time, the current implementation carries a number of limitations that should be acknowledged. The 361ms latency, while manageable for certain musical contexts, remains above what tightly coupled interactive performance typically requires. The output pipeline has been identified as the primary bottleneck and has not yet been fully optimised. The model itself was trained on a relatively narrow corpus of piano music, which likely shapes and limits the stylistic range of its generative output. The correlation evaluation, based on 16 generation cycles and 100 input-output pairs, is a reasonable starting point but a larger measurement dataset would strengthen the statistical claims.

Several future directions for this work are already taking shape. On the technical side, optimising the output pipeline is our most immediate priority, since this is where the largest latency gains are likely to come from. Running multiple simultaneous instances of the external would allow more complex polyphonic textures and open up ensemble performance scenarios. Model fine-tuning on more diverse musical material, or experimenting with alternative transformer architectures, may also broaden the system's musical range.

On the research side, the input-output correlation framework introduced here could be developed further as a general evaluation tool for interactive AI music systems. The three musical dimensions assessed here, pitch, rhythm, and dynamics, are a starting point rather than a complete picture. For pitch correlation in particular, the current evaluation uses a single ±12-semitone threshold against a 19.5% random baseline; future work could report at multiple thresholds simultaneously (±2, ±5, ±7 semitones), corresponding to musically meaningful intervals such as chromatic neighbours, perfect fourths, and perfect fifths, each with their own analytically derived baselines (3.9%, 8.6%, and 11.7% respectively). This multi-threshold approach would give a more granular picture of harmonic coherence and make lift ratios comparable across different thresholds and interaction styles. Beyond pitch, future work might extend the framework to phrase-level structure, assessing whether the AI responds to melodic contour or cadential patterns rather than individual notes and to timbral response for setups involving synthesis or sound design parameters. Expanding the dataset across multiple performers and musical styles would also strengthen the statistical claims and assess how correlation characteristics vary with different playing contexts.

7 CONCLUSION

In this paper, we introduced a real-time MIDI transformer generative system implemented as a Pure Data external, designed to bring transformer-based AI music generation into the dataflow programming environments that many musicians already work in. The system addresses an ongoing gap between the computational demands of transformer inference and the low-latency requirements of live musical performance, through a multi-threaded architecture, a dual-buffer management system, and immediate event-driven generation triggering. To our knowledge, this represents the first implementation of a real-time ONNX-based transformer model running natively inside Pure Data. The architectural approach appears to offer several practical advantages for live musical interaction: temporal precision through immediate input capture, musical coherence through complete event processing before model input, and stable asynchronous operation that keeps the real-time input thread unblocked during inference.

The open-source release gives future work in AI and music creativity something concrete to build on, and we hope it demonstrates that this kind of transformer-based generation can work within accessible open-source programming environments that musicians already use.

AUTHOR DECLARATIONS

This research was supported by internal institutional funding by Aalto Creatives within the SOPI and Engineering Psychology research groups at Aalto University. The research did not receive external commercial sponsorship. The authors declare no financial or non-financial conflicts of interest. The work received ethical review approval from our institutional Aalto University School of Art, Design and Architecture review board. This work was supported by the European Research Council (ERC) under the European Union's Horizon Europe research and innovation programme, AmplifAI (grant agreement No. 101217557), and by the European Union within the Horizon Europe research and innovation programme, XTREME (grant agreement No. 101136006). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union, the European Commission, or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. As the research did not involve experiments with human participants or animals, the review confirmed that no further ethical constraints applied to the conduct of this work. All datasets used (ADL and MAESTRO) were drawn from publicly available sources; licensing terms and stated usage conditions were reviewed and followed to the best of our understanding.

REFERENCES

Agostinelli, A., Denk, T. I., Borsos, Z., Engel, J., Verzetti, M., Caillon, A., Huang, Q., Jansen, A., Roberts, A., Tagliasacchi, M., et al. (2023). Musiclm: Generating music from text. *arXiv preprint arXiv:2301.11325*.

Bigo, L., Lascabettes, P., Nika, J., and Chemillier, M. (2022). Somax2: A modular and reactive co-improvisation system. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

Bryan-Kinns, N., Zhang, B., Zhao, S., and Banar, B. (2024). Exploring variational auto-encoder architectures, configurations, and datasets for generative music explainable ai. *Machine Intelligence Research*, 21(1):29–45.

Caillon, A. and Esling, P. (2021). Rave: A variational autoencoder for fast and high-quality neural audio synthesis. In *Proceedings of the International Conference on Digital Audio Effects (DAFx)*.

Caillon, A. and Esling, P. (2022). nn~: a lightweight c++ library for neural network inference in the audio domain. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

Casini, L., Jonason, N., and Sturm, B. L. T. (2024). Sparks of musical agi? challenges and perspectives in music co-creation with llms. In *Proceedings of the 5th Conference on AI Music Creativity*. Zenodo.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q., and Salakhutdinov, R. (2019). Transformer-XL: Attentive language models beyond a fixed-length context. In Korhonen, A., Traum, D., and Màrquez, L., editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Dong, H.-W., Chen, K., Dubnov, S., McAuley, J., and Berg-Kirkpatrick, T. (2023). Multitrack music transformer. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE.

Drymonitis, A. (2023). [neuralnet]: A pure data external for the creation of neural networks written in pure c.

Engel, J., Agrawal, K. K., Chen, S., Gulrajani, I., Donahue, C., and Roberts, A. (2019). Gansynth: Adversarial neural audio synthesis. *arXiv preprint arXiv:1902.08710*.

Engel, J., Hantrakul, L., Gu, C., and Roberts, A. (2020). Ddsp: Differentiable digital signal processing. *arXiv preprint arXiv:2001.04643*.

Feng, Y., Sahin, E., and Casey, M. A. (2024). Phrase-level symbolic music generation. In *Proceedings of the 5th Conference on AI Music Creativity*. Zenodo.

Ferreira, L. N., Lelis, L. H., and Whitehead, J. (2020). Computer-generated music for tabletop role-playing games.

Haki, B., Nieto, M., Pelinski, T., and Jordà, S. (2022). Real-time drum accompaniment using transformer architecture. In *Proceedings of the 3rd Conference on AI Music Creativity*. AIMC.

Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang, C.-Z. A., Dieleman, S., Elsen, E., Engel, J., and Eck, D. (2019). Enabling factorized piano music modeling and generation with the MAESTRO dataset. In *International Conference on Learning Representations*.

Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., Dai, A. M., Hoffman, M. D., Dinculescu, M., and Eck, D. (2018). Music transformer.

Huang, Q., Park, D. S., Wang, T., Denk, T. I., Ly, A., Chen, N., Zhang, Z., Zhang, Z., Yu, J., Frank, C., et al. (2023). Noise2music: Text-conditioned music generation with diffusion models. *arXiv preprint arXiv:2302.03917*.

Huang, Y.-S. and Yang, Y.-H. (2020). Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions. In *Proceedings of the 28th ACM international conference on multimedia*, pages 1180–1188.

Karystinaios, E. and Widmer, G. (2023). Notochord: A flexible probabilistic model for real-time midi performance. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*.

Mitra, R. and Zualkernan, I. (2025). Music generation using deep learning and generative ai: a systematic review. *IEEE Access*.

Payne, C. (2019). Musenet. https://openai.com/blog/musenet.

Qin, Y., Xie, H., Ding, S., Tan, B., Li, Y., Zhao, B., and Ye, M. (2023). Bar transformer: a hierarchical model for learning long-term structure and generating impressive pop music. *Applied Intelligence*, 53(9):10130–10148.

Roads, C. (1996). *The computer music tutorial*. MIT press.

Rowe, R. (1992). *Interactive music systems: machine listening and composing*. MIT press.

Scarlatos, A., Wu, Y., Simon, I., Roberts, A., Cooijmans, T., Jaques, N., Tarakajian, C., and Huang, A. (2025). Realjam: Real-time human-ai music jamming with reinforcement learning-tuned transformers. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, pages 1–9.

Shih, Y.-J., Wu, S.-L., Zalkow, F., Müller, M., and Yang, Y.-H. (2022). Theme transformer: Symbolic music generation with theme-conditioned transformer. *IEEE Transactions on Multimedia*, 25:3495–3508.

Tahiroglu, K. and Wyse, L. (2024). Latent spaces as platforms for sonic creativity. In *Proceedings of the 16th International Conference on Computational Creativity, ICCC*, volume 24.

Tahiroğlu, K. and Kastemaa, M. (2022). Augmented granular synthesis method for gan latent space with redundancy parameter. In *Proceedings of the 3rd Conference on AI Music Creativity*. AIMC.

Tahiroğlu, K., Kastemaa, M., and Koli, O. (2021). Ai-terity 2.0: An autonomous nime featuring ganspacesynth deep learning model. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, Shanghai, China.

Tremblay, P. A., Roma, G., and Green, O. (2021). The fluid corpus manipulation project. *Computer Music Journal*, 45(2):14–27.

Wang, Z., Zhang, K., Wang, Y., Zhang, C., Liang, Q., Yu, P., Feng, Y., Liu, W., Wang, Y., Bao, Y., et al. (2022). Songdriver: Real-time music accompaniment generation without logical latency nor exposure bias. In *Proceedings of the 30th ACM International Conference on Multimedia*, pages 1057–1067.

Wu, Y., Wang, M., Lei, H., Brade, S., Blanchard, L., Wu, S.-L., Courville, A., and Huang, A. (2025). Streaming generation for music accompaniment. *arXiv preprint arXiv:2510.22105*.

Yang, L.-C. and Lerch, A. (2020). On the evaluation of generative models in music. *Neural Computing and Applications*, 32(9):4773–4784.

Zheng, S. J., Xambó Sedó, A., and Bryan-Kinns, N. (2025). Exploring gestural affordances in audio latent space navigation. *Frontiers in Computer Science*, 7:1575202.