Real-Time Language Model Jamming: A Case Study for Live Music Accompaniment Generation

Abstract—**Language models (LMs) have become one of the most prominent paradigms in modern generative modeling. While making them faster has been the main focus of real-time deployment, speed alone is not enough. Many real-world applications, such as synchronized translation and voice synthesis, also require precise alignment between generation and external signals, both in terms of generation content and timing. We refer to this problem as *frame-synchronous streaming inference*. To address it, we present StreamMUSE, an inference system that performs LM generation in response to an external signal stream within a client–server architecture. The client continuously sends high-frequency inference requests based on the most recent inputs and receives outputs synchronized to the external clock, while the server executes model inference. We demonstrate the framework through a live music accompaniment task, showing how real-time synchronization can be achieved across different deployment environments with varying round-trip latencies. We further model the relationship between system hyperparameters and round-trip latency, and evaluate how different environments affect optimal configurations to achieve real-time performance. Experimental results show a consistent correspondence between system real-time performance and music quality, demonstrating the effectiveness of the proposed framework.**

I. Introduction

Language models (LMs) have become the foundation of modern generative systems across modalities, from text to speech and music. As their applications expand into interactive and real-time domains, such as live conversation, synchronized translation, and music editing, the primary goal has often been to make inference faster [1], [2], [3]. Yet speed alone does not tell the full story. Consider the problem of improvising music with other musicians: the system must not only generate notes quickly, but also update its context frequently to ensure high-quality predictions and stay aligned with the shared musical clock. This highlights a broader challenge in real-time generative modeling—producing outputs that are *synchronized in time* and *coherent in content* with continuously evolving external signals.

In this work, we frame this challenge as *frame-synchronous streaming inference*, where a language model $P_\theta(y|x)$ must generate an output sequence y that evolves in real time with an external signal stream x. The generation proceeds at a fixed frame rate, synchronized to physical time, so that at each frame t, the model produces y_t conditioned on the most recent external information $x_{<t}$ and its own generation history $y_{<t}$. Existing approaches, however, typically operate at much coarser or irregular temporal resolutions. They either assume long frames to predict low-resolution signals (e.g., 2-second chunks [4]) or use dynamically triggered updates tied to user interactions (e.g., chatbots that respond after full sentences [5], [6]). In contrast, frame-synchronous streaming inference targets short frame intervals (e.g., within 200 ms), comparable to the smallest perceptual or structural unit in music or speech (such as a tatum or syllable), where precise timing and synchronization are essential.

In this paper, we present StreamMUSE, a system for frame-synchronous streaming inference. We take real-time music accompaniment generation as a case study, where the model generates accompaniment conditioned on a user-input melody that serves as the external signal. StreamMUSE operates in a client–server architecture: the client continuously sends high-frequency inference requests based on the latest user input, and the server performs language model inference and returns generation results aligned to the music stream. The system is characterized by two key parameters: the *inference interval*, which determines how frequently requests are sent, and the *generation length*, which specifies how long each request generates output. These hyperparameters jointly balance responsiveness and reliability under stochastic round-trip latency, which depends on model inference time and network conditions between client and server.

To analyze this trade-off, we develop a latency-aware formulation that quantifies the feasible configurations of these parameters under varying latency conditions. A straightforward intuition is that real-time inference can be achieved by minimizing the inference interval and setting the generation length just long enough to cover each frame. In practice, however, random latency fluctuations make hyperparameter configuration non-trivial: overly short intervals can cause overlapping requests that stall the system, while slightly longer generation lengths introduce temporal redundancy that provides a backup against delayed or failed requests. We further examine how an increment of music tempo—the innate music concept that defines the frame rate—tightens these constraints, sometimes even making real-time scheduling infeasible.

We train a real-time accompaniment generation model as the backbone LM and evaluate StreamMUSE across three environments—local, local-server, and remote-server—with varying network latencies and inference speeds. Experimental results show that system behavior aligns well with our model for hyperparameter selection, and that the system responsiveness is strongly correlated with music-quality metrics. The experiments further confirm that, when feasible under given latency conditions, more frequent inference requests yield performance closest to the offline baseline, demonstrating the importance and necessity of high-resolution streaming inference.

II. Background and Related Work

This section surveys developments in LLM inference, music generation, and real-time systems, highlighting this critical disconnect.

A. Language Model Inference

Significant research efforts have been dedicated to optimizing and accelerating large language model (LLM) inference, primarily following two complementary technical pathways. The first focuses on architectural innovations, such as transforming dense models into sparse Mixture-of-Experts (MoE [7]) structures, which preserve total parameter count while substantially reducing computational load per input token. The second approach retains the algorithmic foundation while optimizing resource utilization at the framework level—exemplified by systems like vLLM [8] and SGLang [9], which introduce advanced techniques such as continuous batching, paged KV cache management, and adaptive scheduling. In hard real-time task scenarios such as automotive intelligent software [10], systems must further incorporate task prioritization and resource management on top of model-level optimizations, ensuring critical functions (e.g., braking decisions) receive prioritized inference resources. In summary, existing optimizations are primarily designed for general-purpose language models, whose token-sequential generation process and heavy reliance on statistical dependencies make them difficult to directly apply in edge scenarios with strict requirements for real-time performance, determinism, and task criticality.

B. Machine Learning for Music Generation

Music has long been intertwined with real-time interaction, dating back to early research on score following [11], [12], timbre recognition [13], [14], and other interactive music systems that require instantaneous computational responses [15], [16], [17]. Accompaniment generation, a representative task in music AI, is particularly well suited for real-time musical interaction due to its inherently responsive and continuous nature.

Deep learning has significantly advanced symbolic music generation, with models such as Music Transformer[18] and Multitrack Music Transformer[19] excelling at modeling long-term structure and multi-instrument arrangements. Several studies have further explored controllable symbolic generation, such as melody-conditioned accompaniment. AccoMontage [20] retrieves and adapts stylistic phrase representations from a precompiled corpus; Whole-Song Hierarchical Generation [21] generates accompaniment hierarchically from global structure to detailed parts; and the Anticipatory Music Transformer [22] improves quality through a look-ahead design that explicitly violates real-time assumptions. These methods produce musically coherent and controllable results, yet real-time generation remains challenging because they rely on offline processing and full-sequence context.

C. Real-Time Accompaniment Generation System

Despite major advances in deep learning for symbolic music modeling, its integration into real-time interactive systems has been relatively limited. Only a few recent studies have explored this direction, yet their focus largely lies on algorithmic design—particularly in reinforcement learning—rather than on the system-level challenges of achieving stable, low-latency interaction. The RL-Duet framework [23] formulates real-time accompaniment as a sequential decision-making process using reinforcement learning, enabling the agent to respond instantaneously to a performer's input; however, its discrete action formulation leads to coarse timing and limited expressiveness. Similarly, Bach-Duet[24] adopts a reinforcement-learning paradigm but focuses on phrase-level musical responses rather than fine-grained, continuous control. ReaLJam[25] introduces a reinforcement-learning-tuned Transformer for interactive jamming with predictive scheduling, yet its design remains centered on chord- or measure-level anticipation. The Jam Bot [26] enables human–AI free improvisation by adapting symbolic music language models, but its architecture is artist-specific and performance-oriented, relying on fine-tuned stylistic data and handcrafted interaction strategies that limit generalization to broader real-time or multi-user scenarios. Overall, these works represent valuable early steps but remain coarse-grained in temporal resolution and system integration. In contrast, our work focuses on the tackling the system problem—building a responsive, low-latency, and robust real-time generation framework—addressing the fundamental gap between algorithmic sophistication and deployable real-time musical interaction.

III. Model Overview

A. Statistical Model for Real-time Accompaniment Generation

In this study, we formulate the real-time interactive accompaniment task as a *conditional sequence modeling problem*. The core design principle, crucial for low-latency collaboration, is the online constraint: upon each generation request, the model must only generate new accompaniment tokens conditioned on the observed melody history and all previously generated accompaniment. This process is strictly causal and permits no lookahead into the user fed future melody.

Considering the incremental and rhythmic nature of real-time music generation, in this work we discretize time into uniform temporal steps called *ticks*, as the finest resolution considered. Its size is configurable, though in our experiments a tick is fixed at 1/4 beat (1/16 note), its wall clock time duration τ_{tick} depends on the music speed (commonly measured in BPM, or Beats Per Minute). E.g., for a song of 120 BPM, τ_{tick} would be 125ms.

Formally, let $\mathbf{M}[i:j]$ denote the melody token sequence and $\mathbf{A}[i:j]$ denote the accompaniment token sequence starting at tick i and ending at tick j, the model estimates the conditional distribution

$$p(\mathbf{A} \mid \mathbf{M}) \;=\; \prod_{t=0}^{T-1} p\big(\mathbf{A}[t] \mid \mathbf{A}[0:t-1],\, \mathbf{M}[0:t-1]\big). \quad (1)$$

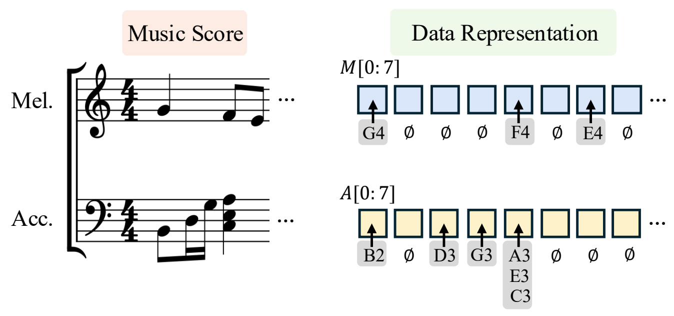


Fig. 1: Sample MIDI tokenization

In other words, for tick t, the model should predict $\mathbf{A}[t]$ prior to observing this segment melody $\mathbf{M}[t]$, so that $\mathbf{A}[t]$ can be produced in time to accompany it. A tokenization method that supports the real-time setting and model architecture is introduced later in this section.

B. Data Representation

The data representation described above is a conceptual abstraction, to support real-time generation, we employ a more detailed and realistic music data encoding scheme. Based on the tick-based granularity introduced earlier, we represent the melody and accompaniment as two aligned sequences of *frames*:

$$\mathbf{M} = (\mathbf{M}[0], \dots, \mathbf{M}[t-1]),$$
$$\mathbf{A} = (\mathbf{A}[0], \dots, \mathbf{A}[t-1]).$$

To enable efficient autoregressive prediction, these two sequences are combined using an *interleaving* technique into a single input sequence

$$\mathrm{InterL}(\mathbf{M}, \mathbf{A}) = (\mathbf{A}[0], \mathbf{M}[0], \dots, \mathbf{A}[t-1], \mathbf{M}[t-1]).$$

Given the polyphonic nature of music, both a melody frame ($\mathbf{M}[t]$) and an accompaniment frame ($\mathbf{A}[t]$) are designed as ordered subsequences that encapsulate all note events occurring at tick t. This allows a single frame to summarize multiple simultaneous notes (or silence). For example, an accompaniment frame can represent three simultaneous notes forming a chord. Fig. 1 illustrates such tokenization. Here, the 2 beats of sheet music shown on the left side translate into 8 ticks on the right, where internal representations like “B2” define the pitch of notes. Each blue/yellow block in the figure represents a melody/accompaniment frame.

Note that based on the polyphony encountered in our training and testing music datasets, we empirically set the maximum number of note events per frame to 4, which is easy to extend if necessary.

C. Model Architecture and Training

Our model itself is a Transformer-based, autoregressive decoder conditioned on the interleaved melody and accompaniment sequence. The architecture is built around a three-module structure—a Local Encoder, a Global Predictor, and a Local Decoder—designed to efficiently process the structured music representation [27]. The Local Encoder encodes notes starting at the same tick into a single token. The Global Predictor then processes these aggregated vectors across the entire history to capture global musical and rhythmic context and predict high level representation of the next token. Finally, the Local Decoder generates the new frame’s subsequence autoregressively, conditioned on the Global Predictor’s output for the current time step.

The loss function used for training this model is the standard cross-entropy loss based on Equation 1. During training, the training data sequences start with the accompaniment token $\mathbf{A}[0]$. Since our model is trained with teacher forcing, the resulting discrepancy between training and inference—known as exposure bias—is an inherent challenge in this streaming setting.

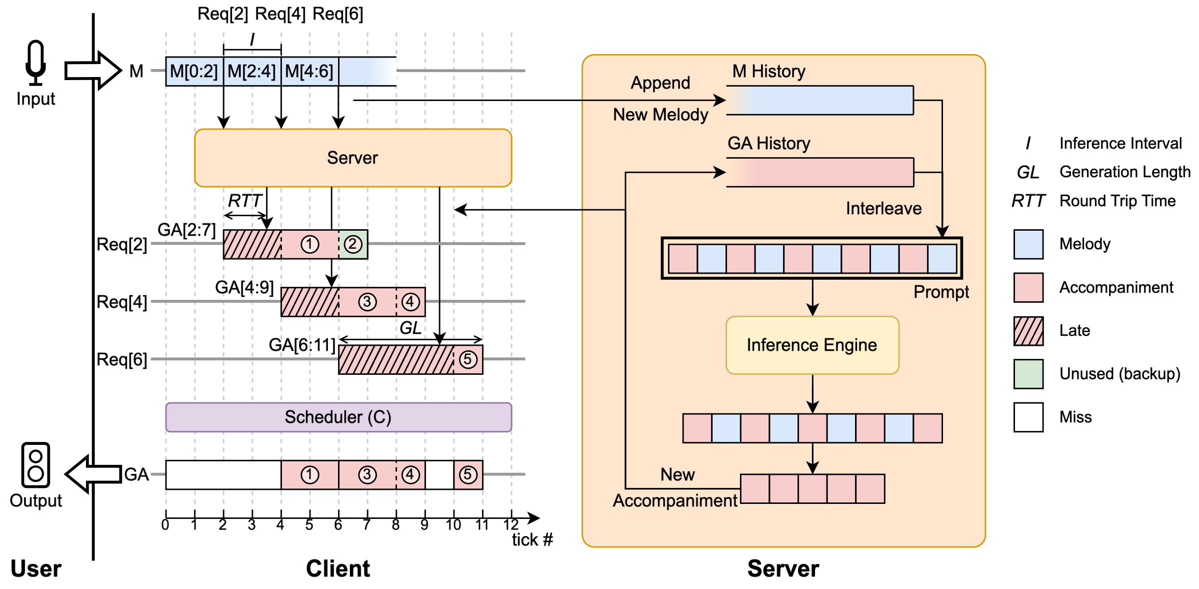
D. StreamMUSE Architecture Overview

Finally, we assemble the above building blocks and give a sketch of the architecture of our proposed StreamMUSE system. While the above model allows incremental accompaniment generation, achieving real-time accompaniment is far from straightforward. Live accompaniment is highly sensitive to latency and BPM (A higher BPM accelerates the musical rhythm, leading to more rhythmically discordant beats under identical latency conditions.) Only when the accompaniment closely follows the performer’s melody does the overall performance sound natural. Yet, the round trip time (RTT) of fulfilling each request due to network latency and server side inference is significant, making it difficult for the system to produce perfectly synchronized accompaniment in real time.

To enable StreamMUSE’s working with diverse hardware and service settings, we adopt the common client-server design, as illustrated by Fig. 2. Such a distributed architecture also decouples latency-sensitive user interaction from computation-intensive model inference.

The client, which takes user input MIDI melody, sends accompaniment generation requests at a certain frequency, again defined by tick intervals (I ticks). The server processes a request, generating an interleaved sequence (using the InterL function), whose size is specified by the generation length parameter GL. It is desirable to have a GL long enough to produce accompaniment sequences overlapping in time, so that there are backup segments in case of network or inference latency jitters. The accompaniment component of the generated sequence, GA, is then returned to the client, which coordinates the responses from multiple outstanding requests and schedules the accompaniment playback. Both the ground-truth (user-fed) melody and the GA results are maintained by the server side for subsequent inferences.

Table I summarizes the key definitions introduced, to be used throughout the paper. In next section, we revisit Fig. 2 and give detailed description of the design of our client and server. In particular, how they work together to automatically search for optimized I and GL parameter values under a given hardware/service setup.



(a) Request timeline (b) Server workflow

Fig. 2: StreamMUSE architecture. Here the left side gives a sample timeline of 3 requests, issued at the onset of tick 2, 4, and 6, respectively. The right side gives a zoomed-in view of the “server” box shown within the timeline.

TABLE I: Definitions

Symbol	**Description**
$\mathbf{M}, \mathbf{A}$	Sequences of melody and accompaniment frames
BPM	Beats per minute
tick	Minimum work unit in StreamMUSE’s processing, set as a fraction of a beat
τ_{tick}	Tick duration in milliseconds
$\text{InterL}(\mathbf{M}, \mathbf{A})$	Interleaved melody–accompaniment token sequence
$\mathcal{L}$	Cross-entropy loss function
I	Inference Interval
GL	Generation Length
GA	Generated Accompaniment
Req	Request
RT_{tick}	Round trip time in tick unit
RTT	Round trip time in milliseconds

IV. StreamMUSE Design and System Modeling

A. Client Design

The client employs a multi-threaded design and an asynchronous scheduling mechanism that decouples real-time playback from network communication. The Main Thread coordinates timing and playback scheduling, the Input Thread captures user melody input in real time, and the Inference Thread manages communication with the server. At each time interval (typically consisting of several ticks), the client packages the latest melody segment and sends it to the server for inference. It then receives a generated accompaniment of length GL from the server for playback scheduling.

In practical network modeling, we have observed that network latency follows a long tail distribution, meaning a small fraction of requests exhibit round-trip times (RTT) significantly higher than the average. To address this, we implemented a tick aligned data structure and introduced a backup mechanism as a safety buffer. The generated accompaniment (GA) returned by each server is stored in this data structure for real-time scheduling. By setting the generation length (GL) to be longer than the inference interval (I), an overlap is created between consecutive GAs. This overlapping portion serves as a backup left by the previous request and will be overwritten by data from a new, on time request. If a new request is delayed, the backup is scheduled in real-time to maintain musical continuity. A missing accompaniment event occurs only when a request is delayed and no backup accompaniment is available in the system.

In the sample scenario shown in Fig. 2(a), I is set to 2 and GL is set to 5 ticks (therefore 9 frames in the interleaved sequence). After experiencing an RTT of 1.5 ticks, Request-1 received its return value and wrote it to GA[2:6]. Due to the upward rounding of ticks (where a tick is the smallest scheduling unit of the accompaniment), the shadow of GA is already too late to be scheduled. GA① can be directly scheduled in real-time, while GA② serves as a backup for subsequent accompaniment. Since Request-2 returns as expected, the generated accompaniment is overwritten, and the newly returned GA③ will be scheduled, with GA④ acting as the backup. However, when Request-3 is delayed, the backup from the last request GA④ will be scheduled; however, when the delay exceeds its coverage, no available GA remains for scheduling, resulting in a miss until Request 3 returns.

The backup mechanism effectively mitigates the impact of network tail latency. However, having excessively long backup generation is clearly not advantageous. Although it can reduce the risk of missing accompaniment, it also increases the inference latency for all requests, potentially compromising the system's real-time constraints. Hence, the goal of our real-time system is twofold: (1) to ensure musical continuity by minimizing the number of missing events (the white "hole" at tick 9 in Fig.2), and (2) to satisfy the real-time constraint by reducing reliance on backup accompaniment. Later in this section, we further formalize the real-time behavior of StreamMUSE.

B. Server Design

The core of the server is an inference engine built on the generative model described in Section III. As shown in Fig. 2 (b), during online generation, the server maintains a tick aligned data structure to store the user's historical generated accompaniment and real melodic context information. Upon receiving the real-time melody from the client, the server appends it to the historical melody data. The accompaniment and melody at the same tick are then interleaved to form a new sliding window, which is fed into the model as a unified melody–accompaniment sequence prompt. The model infers and generates the next segment of the melody–accompaniment (9 frames in this example), where the 5 frames/ticks of accompaniment result are returned to the client, as well as saved as accompaniment history.

After inference, the server separates the generated melody and accompaniment, discards the generated melody, retains the generated accompaniment, and transmits it to the client. This mechanism minimizes data transmission between client and server while maintaining real-time updates of the user's melody, ensuring both real-time performance and the accuracy and contextual relevance of the generated results.

C. Request Response Time Modeling

As previously discussed, RTT is an inherent system latency that fundamentally determines the theoretical upper bound of real-time performance in online music generation. To model RTT, our analysis decomposes it into two core components: (i) inference latency (IL), which refers to the computational time required on the server side to complete the music generation; and (ii) network latency (NL), which encompasses the communication overhead incurred during end to end transmission of the request and response data.

To model IL, the key relevant factor is the model's autoregressive generation length (GL), as a request's RTT can be formally expressed as

$$RTT(GL) = IL(GL) + NL$$

Across multiple hardware and inference service setups, we sweep the typical values GL and record IL to analyze their relationship. Through our detailed breakdown, we find that in the current three module structure, IL is dominated by the local decoder that autoregressively generates frames. Within the computational complexity of the local decoder, the leading term is determined by the attention mechanism, whose cost scales quadratically with GL. Accordingly, we model IL as a quadratic function of GL with an additive Gaussian noise term:

$$IL = f(GL) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2),$$

where ϵ represents the stochastic fluctuations in inference latency caused by system load and other runtime uncertainties.

To model NL, in a local setup where the client and server run on the same host, there is no network communication, so we set NL to 0; with the other network setups, NL fluctuates around a constant value due to fixed communication overhead and unstable wireless links. To capture the heavy tailed behavior of network latency, we replace the Gaussian noise with a shifted power law (Pareto) noise term:

$$NL = \nu + (\xi - x_{\min}), \quad \xi \sim \text{Pareto}(x_{\min}, \alpha),$$

where ν is a constant representing the fixed communication delay, $x_{\min} > 0$ is the scale (minimum) parameter, and $\alpha > 1$ is the tail index controlling the heaviness of the tail. The probability density function (PDF) of ξ is given by

$$g_\xi(x) = \frac{\alpha\, x_{\min}^{\alpha}}{x^{\alpha+1}}, \quad x \ge x_{\min}$$

This formulation ensures that the noise term $(\xi - x_{\min})$ has a minimum value of zero, allowing NL to fluctuate around ν while exhibiting occasional extreme latency spikes consistent with heavy tailed network behavior.

Putting the above together, we write RTT in explicit form as

$$RTT = f(GL) + \nu + (\xi - x_{\min}) + \epsilon$$

D. Model-based StreamMUSE Parameter Configuration

Finally, we come to the key problem of a real-time interactive music accompaniment system, to balance generative quality, interactive responsiveness, and the physical constraints of computation and networking. To this end, our request response time modeling given above allows us to move beyond ad-hoc parameter tuning, adopting instead a formal methodology that derives the feasible space of real-time safe configurations. In this manner, under different hardware, network, and inference service configurations, StreamMUSE automatically identifies the optimal operating point to maximize interactivity and reliability.

The derivation depends on relating the system's temporal latency (in milliseconds) to the musical "tick" (our discrete time unit). We first introduce a key parameter:

- τ_{tick} (Time per Tick): The duration of a single tick in seconds. A 16th note resolution (4 ticks per beat), the time per tick is derived from BPM:

$$\tau_{tick} = \frac{60 \times 1000}{4 \cdot BPM} (\text{ms/tick})$$

For this constraint analysis, we simplify the RTT model from Section 4.2 by combining the 95th percentiles of γ (Constant term of f(GL)), ν, $(\xi - x_{\min})$, and ϵ into a single constant term

c for tractability. This yields the following approximation for the 95th percentiles tick level response time:

$$Q_{0.95}(\{\gamma, \nu, \xi - x_{\min}, \epsilon\}) \leq c$$

$$RT_{tick}^{95} \approx \frac{f(GL) + c}{\tau_{tick}} \tag{2}$$

With these variables defined, we establish two fundamental constraints for real-time operation:

1) Real-Time Safety Constraint: The client should ideally receive the server generated accompaniment before the next interval begins.

$$\lceil RT_{tick}^{95} \rceil \leq I$$

2) Playback Continuity Constraint: The duration of the generated accompaniment must be designed to accommodate outlier RTT instances present in the long tail segment of the latency distribution:

$$GL \geq I + \lceil RT_{tick}^{95} \rceil$$

Rearranging this inequality to solve for I, we get the form used in our final derivation:

$$I \leq GL - \lceil RT_{tick}^{95} \rceil$$

By combining our two constraints, we derive a single, two sided inequality that *defines the valid range for* the *Inference Interval (I)* and substituting our RTT model $f(GL)$ into it gives the fully expanded constraint:

$$\frac{f(GL) + c}{\tau_{tick}} \leq I \leq \frac{\tau_{tick} GL - f(GL) - c}{\tau_{tick}} \tag{3}$$

In our experimental environment, the behavior of $f(GL)$ can be accurately modeled using a quadratic function. For any given network condition, we first determine the simulated RTT function through benchmarking. Then, for a specified BPM (which defines the value of τ_{tick}), we iterate over a range of possible Generation Lengths (GL). For each GL, we validate the feasibility condition by computing the lower and upper bounds from Equation 3. If the lower bound is less than or equal to the upper bound, we proceed to calculate the valid integer range for I (i.e., $\lceil \text{lower bound} \rceil \leq I \leq \lfloor \text{upper bound} \rfloor$). Any integer I within this range forms a stable (GL, I) configuration pair. Through this search process, we can identify all valid configuration pairs, thereby delineating the feasible search space. This space can then be carefully evaluated to select configurations that optimize other metrics—such as choosing the smallest I for optimal responsiveness, or selecting specific configurations to reduce server load.

Corollary (Feasibility Condition). A necessary precondition for system feasibility is that the generation length must be at least twice the 95th percentile tick level response time:

$$GL \geq 2 \lceil RT_{\text{tick}}^{95} \rceil$$

Rearranging this formulation yields the equivalent real-time constraint:

$$RTT(GL) \leq \frac{\tau_{\text{tick}}}{2} GL$$

TABLE II: Specifications of three target settings

Setting	Client	Server	Network
Local	Hyperstack server (RTX A6000)	Hyperstack server (RTX A6000)	IPC
Local-server	Macbook (M1 Pro)	PC (RTX 3090)	6Gbps WLAN
Remote-server	Macbook (M1 Pro)	Hyperstack server (A100)	1Gbps WAN

which indicates that satisfying the real-time requirement is equivalent to ensuring that the $RTT(GL)$ function remains entirely below the linear upper bound with slope $\frac{\tau_{\text{tick}}}{2}$.

V. Evaluation

A. Experimental Setup and Model Preparation

Implementation We build our sample real-time music accompaniment system via Python with more than 5000 lines of codes, containing the full implementation of client-server architecture. For client side, we use multi-threaded mechanism to implement its different component; for server side, we select Transformers [28] library from HuggingFace as the inference engine, and enable KVCache optimization [29] to avoid redundant computation of the autoregressive generation of local decoder. Client and server use TCP protocol to communicate with each other when network is needed.

Testbed We analyze the performance of our system under three different settings:

- **Local** Client and server reside on the same device, directly communicating with each other via inter-process communication (IPC).
- **Local-server** Client and server are hosted on separate devices within the same wireless local area network (WLAN).
- **Remote-server** Client resides in a home network, while the server is deployed on a public cloud platform (e.g., Hyperstack). Communication between them is established via a wireless wide area network (WAN).

For the listed three settings, we summarize the hardware and network specifications in Table II.

Baselines For the real-time performance, since we are the first to rigorously model real-time interactive transformer system, we mainly compare the performance of different StreamMUSE's configs within the search space to demonstrate the effectiveness of our parameter selection. We also compare the music quality of StreamMUSE with generated music from the offline version of our accompaniment generation model, which is used as the theoretical upper bound for interactive music quality.

Training Details We use the POP909 dataset [30] for training, consisting of 909 pop music pieces. The model is trained using the standard cross-entropy loss on the interleaved token sequence. We also apply pitch shifting for data augmentation, where the shift amount is uniformly sampled from the allowed range (clipped to $[\mathbf{-5}, \mathbf{6}]$ semitones) for both melody and accompaniment tracks, and use gradient clipping to stabilize training. For evaluation, we primarily utilize the Small model (0.12B parameters). Its three-module architecture consists of a 12-layer main module alongside a 3-layer local encoder and a 3-layer local decoder.

Evaluation Song Selection Our conditional sequence modeling task inherently requires music data where melody (mel) and accompaniment (acc) are cleanly separated into distinct tracks. However, most publicly available MIDI datasets do not provide this essential separation. To address this common challenge, we adopt the standard industry practice of skyline extraction. This method heuristically extracts the most prominent melodic line from a polyphonic music track, ensuring the necessary separation for our model's conditioning. For our experiments, we constructed a dedicated test set with 64 music pieces by sampling 30 from AccoMontage and 34 from Pop1k7 [31].

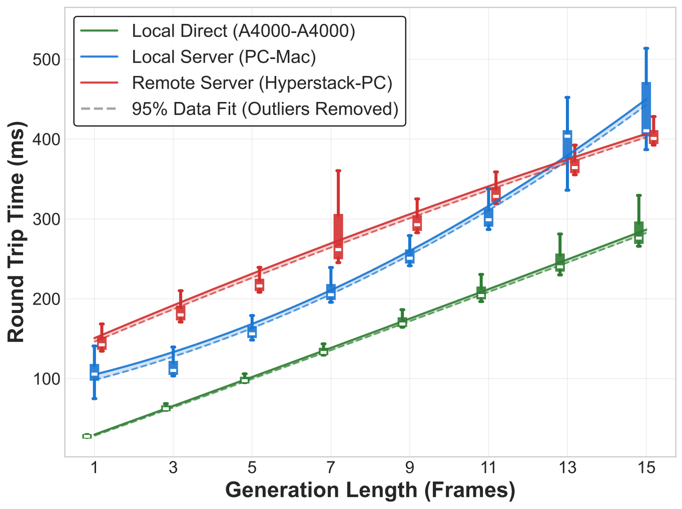


Fig. 3: Comparison between RTT model and real RTT latency

B. RTT Latency Characteristics

To understand the RTT patterns of the *Local Direct*, *Local Server*, and *Remote Server* environments, we profiled them by varying the generation length (GL) and collecting 1,000 RTT samples for each. The resulting distributions are shown in Fig. 3.

Rather than simply validating our model, we use our quadratic RTT model as a tool to characterize and quantify each environment's performance. As shown in Fig. 3, we fit parameters (α, β, γ) to both 100% (solid lines) and 95th percentile (dashed lines) of the data. We use the 95th-percentile fit for our analysis, as it avoids skew from rare outliers and provides the robust RT^{95}_{tick} boundary essential for our safety constraints (Section IV-D). This analysis reveals three distinct profiles:

- Local Direct (Green) serves as our computational baseline, isolating performance from network variability. It exhibits a near-zero base latency (γ) and minimal variance, with RTT dominated by predictable computational cost.
- Remote Server (Red) has the highest base latency ($\gamma \approx 150$ms) due to network distance, but its RTT scales slowly, confirming its powerful GPU. This environment is high-latency but predictable, with stable variance (barring $GL = 7$).
- Local Server (Blue) has a lower base latency ($\gamma \approx 100$ms) than Remote, but its RTT scales more rapidly, overtaking the Remote Server at $GL \approx 13$. This environment is also highly unstable, with RTT variance increasing dramatically with GL.

C. Visualization of Solution Space

After validating the fitting of our RTT model, we use RTT model with the constraints described in section IV-D to search for all possible (I,GL) configuration pairs that fulfill real-time requirements under three settings, which is defined as the solution space of StreamMUSE. After getting the solution space, we plot all the feasible real-time configuration pairs explored by our system under Local, Local-server, and Remote-server settings across different BPMs, as illustrated in Fig. 4. The search space is defined by generation length (GL) and inference interval (I). StreamMUSE constructs the feasible solution space bounded by a solid line (representing the Real-Time Safety Constraint) and a dashed line (representing the Playback Continuity Constraint).

In different settings, variation in hardware and network conditions for interactive system may lead to situations where real-time requirements are violated. For instance, when the RTT exceeds the interval I, the client may issue requests at a faster rate than the server's response capacity, resulting in a mismatch between the returned accompaniment and the user's current playing melody—referred to as C1 violation. Alternatively, if GL is shorter than the sum of RTT and I, user may experience missing intervals of accompaniment during playing—referred to as C2 violation. It is clear from Fig. 4 that StreamMUSE can consistently find solution space in different settings, and the solution space varies across settings. In local setting, since the network latency is completely avoided, solution space can occupy no less than 44% of the total search space. On the contrary, in remote-server setting, over 60% of configurations within the search space exhibit violations of real-time conditions, making it challenging to manually construct a configuration that satisfies real-time requirements. Moreover, the changing of the BPM exacerbates the violation, shrinking the solution space to 71%, 33%, 30% in local, local-server, and remote-server respectively. However, by virtue of accurate modeling and rigorous constraints applied to real-time interactive transformer tasks, the StreamMUSE is capable of stably identifying configurations that fulfill real-time requiements under complex and dynamic conditions.

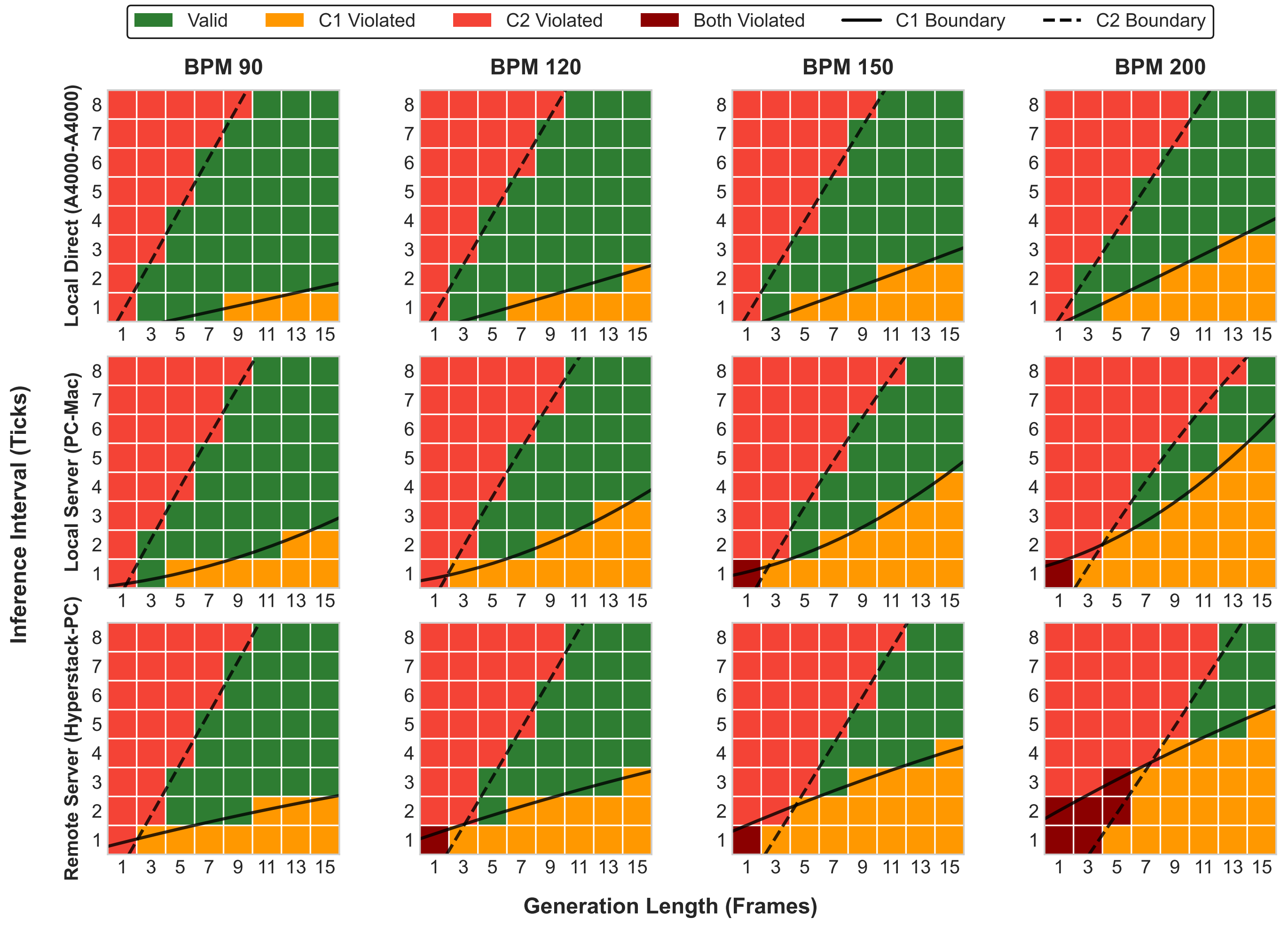


Fig. 4: Visualization of possible configuration pairs in solution space across three settings and four BPMs (Green: valid pair; Yellow: pairs violate C1 constraint; Red: pairs violate C2 constraint; Dark Red: pairs violate both constraint)

D. Key Metrics

We list various metrics here to evaluate the performance of our system in terms of real-time performance and music quality.

Real-time Performance We evaluate the system performance by three metrics: Interaction Success Rate (ISR), Staleness and Weighted Interaction Success Rate (ISR_w).

ISR measures the proportion of time steps where StreamMUSE successfully find a piece of accompaniment to schedule and avoid event missing. First, we define an indicator function $I(t)$ for each time tick t:

$$I(t) = \begin{cases} 1, & \text{if any notes can be scheduled at time } t \\ 0, & \text{otherwise} \end{cases}$$

$I(t) = 1$ means that the system is able to respond to the melody timely, no matter using backup notes or not. Then, the ISR can be defined as the average of this indicator function over the total sequence duration N:

$$ISR = \frac{1}{N}\sum_{t=0}^{N-1} I(t)$$

A higher ISR indicates a more consistently available and responsive system, implying fewer interruptions or missing in the accompaniment generation and scheduling pipeline.
The Staleness metric quantifies the freshness of the scheduled note events. In a low-latency system, a scheduled note should ideally be one of the first tokens produced in the current generation step. We define $S(t)$ as the Staleness Level at time t when a note is scheduled:

$$S(t) = \text{scheduled note's index in its generation sequence}$$

The overall Staleness is the average of $S(t)$ over all successful interaction time steps:

$$Staleness = \frac{1}{|\{t : I(t) = 1\}|}\sum_{t\in\{t:I(t)=1\}} S(t)$$

TABLE III: Experiment Results: Comparative Performance across Real-Time Settings. Results are presented for three settings: (a) Local (low-latency), (b) Local Server, and (c) Remote. Each row represents a configuration defined by the interaction length (I) and generation length (GL). Metrics are divided into Music Quality (JSD, FMD, CR, UR, NLL$_{wavg}$) and System Performance (ISR, Staleness, ISR$_w$). The Offline baseline serves as the upper bound for music quality. Best two scores per column for generated samples are highlighted in bold. $\downarrow$ indicates lower is better, and $\uparrow$ indicates higher is better.

(a) Local

	Music Metrics						**System Metrics**		
(I, GL)	JSD(P) $\downarrow$	JSD(O) $\downarrow$	FMD $\downarrow$	CR $\uparrow$	UR $\downarrow$	NLL$_{wavg}$ $\downarrow$	ISR $\uparrow$	Staleness $\downarrow$	ISR$_w$ $\uparrow$
(1, 3)	**0.2941**	**0.1682**	279.2	**0.8207**	**0.04179**	**1.498**	**0.9952**	**0.001199**	**0.9951**
(2, 5)	0.3206	0.1904	269.5	0.7323	0.1012	1.690	**0.9781**	**0.9969**	**0.9477**
(2, 9)	0.3354	0.3321	329.3	0.4693	0.4409	1.578	0.5795	1.944	0.5443
(4, 5)	0.3831	0.2343	289.3	0.4758	0.3159	1.576	0.6855	1.230	0.6592
(4, 9)	0.3075	0.1834	**259.4**	0.7247	0.1038	1.688	0.9511	2.419	0.8792
(4, 15)	**0.3000**	**0.1800**	290.8	**0.7759**	**0.09315**	1.780	0.9495	3.783	0.8373
(7, 9)	0.3618	0.3492	373.4	0.3413	0.5037	**1.315**	0.6160	2.455	0.5687
(7, 15)	0.3234	0.1976	**262.5**	0.6595	0.1411	1.806	0.8628	4.425	0.7436
Offline	0.1960	0.1111	83.97	0.8257	0.05066	1.586	-	-	-

(b) Local Server

	Music Metrics						**System Metrics**		
(I, GL)	JSD(P) $\downarrow$	JSD(O) $\downarrow$	FMD $\downarrow$	CR $\uparrow$	UR $\downarrow$	NLL$_{wavg}$ $\downarrow$	ISR $\uparrow$	Staleness $\downarrow$	ISR$_w$ $\uparrow$
(1, 3)	0.3456	0.4132	425.7	0.3292	0.5985	1.418	0.3813	**0.5440**	0.3749
(2, 5)	0.3172	**0.1880**	**263.4**	**0.7403**	**0.1082**	**1.273**	**0.9536**	**1.451**	**0.9104**
(2, 9)	0.4876	0.5396	733.1	0.01882	0.9733	**1.079**	0.0373	3.450	0.0333
(4, 5)	0.4096	0.2680	325.3	0.4393	0.3633	1.485	0.5666	1.529	0.5395
(4, 9)	**0.3111**	**0.2000**	**262.9**	**0.6145**	**0.1680**	1.679	**0.7776**	2.977	**0.7052**
(4, 15)	0.3571	0.4665	439.8	0.2218	0.7190	1.401	0.2827	5.281	0.2360
(7, 9)	0.3798	0.3856	389.3	0.2562	0.6166	1.530	0.5131	2.805	0.4681
(7, 15)	**0.3167**	0.2169	354.7	0.4700	0.3516	1.979	0.6384	5.355	0.5316
Offline	0.1960	0.1111	83.97	0.8257	0.05066	1.586	-	-	-

(c) Remote

	Music Metrics						**System Metrics**		
(I, GL)	JSD(P) $\downarrow$	JSD(O) $\downarrow$	FMD $\downarrow$	CR $\uparrow$	UR $\downarrow$	NLL$_{wavg}$ $\downarrow$	ISR $\uparrow$	Staleness $\downarrow$	ISR$_w$ $\uparrow$
(1, 3)	0.3741	0.4041	832.7	0.01673	0.9747	**1.068**	0.01150	**1.000**	0.01114
(2, 5)	0.3093	0.2139	271.1	0.6568	0.1720	1.670	0.8679	**1.471**	0.8280
(2, 9)	0.3222	0.3978	356.4	0.3171	0.6020	1.510	0.4397	2.410	0.4066
(4, 5)	0.4139	0.2876	326.2	0.3792	0.4410	1.480	0.5460	1.573	0.5191
(4, 9)	0.3160	**0.1880**	**257.7**	**0.7010**	**0.1136**	**1.306**	**0.9376**	2.518	**0.8638**
(4, 15)	**0.3044**	**0.1639**	285.4	**0.7939**	**0.05718**	1.351	**0.9894**	3.589	**0.8785**
(7, 9)	0.3935	0.3805	391.1	0.2591	0.6103	1.527	0.5721	2.586	0.5259
(7, 15)	**0.3078**	0.2071	**262.0**	0.6781	0.1408	1.764	0.8709	4.364	0.7522
Offline	0.1960	0.1111	83.97	0.8257	0.05066	1.586	-	-	-

A lower Staleness means the system is quickly identifying and responding accompaniment request, resulting in fresher auditory feedback and better rhythmic alignment. Higher Staleness implies the system is slowly responding to the requests, forcing itself to play stale notes for each request.

The Weighted Interaction Success Rate (ISR$_w$) is a comprehensive metric that integrates both the frequency of successful interactions and response freshness. It is formally defined as:

$$ISR_w = \frac{1}{N} \sum \left(\frac{C - S(t)}{C} \times I(t) \right)$$

where the term $\frac{C-S(t)}{C}$ acts as a staleness penalty for a successful interaction $I(t)$. A higher ISR_W indicates the system is not only good at responding timely, but also delivering fresher responses . ISR_w is crucial for comparing different scheduling strategies in a real-time environment. We set the constant $C = 32$, corresponding to the maximum expected length of a generation sequence or frame.

Music Quality We evaluate the accompaniment generated by our models from two main aspects: the authenticity of the generated music and the harmonic alignment between the accompaniment and the conditioning melody. For the first aspect, we use four metrics: The Fréchet Music Distance (FMD) [32] and Negative Log-Likelihood (NLL). FMD measures the statistical distance between distributions of features extracted from real and generated symbolic music, lower values indicate that the generated samples are closer in distribution to real performances. NLL quantifies the internal fluency and statistical coherence of the accompaniment generated by our real-time system. Lower NLL suggests that the generated music is statistically more consistent and fluent according to the model's learned distribution. Furthermore, we use the Jensen-Shannon Divergence (JSD) to compare the statistical distributions of generated and real accompaniment: JSD(P) (Pitch) quantifies the fidelity of the note vocabulary used, and JSD(O) (Onset) evaluates the accuracy of the rhythmic and temporal patterns; lower values for both indicate better fidelity. For the second aspect, the harmonicity metric evaluates how well the generated accompaniment supports the melody over time. For each melody note, we check all simultaneous accompaniment notes: if none overlap, that duration is counted as "unsupported." When overlap exists, we compute pitch-class intervals relative to the melody and compare them against a designated consonant set (unison, minor/major thirds, perfect fifth, and minor/major sixth). Durations with at least one consonant interval are labeled "consonant," while others are "dissonant." Normalizing these by the total melodic duration yields three distinct ratios: Consonant Rate (CR), Unsupported Rate (UR), and the Dissonant Rate. These ratios reflect how often the accompaniment harmonically supports, fails to reinforce, or conflicts with the melody, respectively. For the purpose of our study, we choose to present the Consonant Rate (CR) and the Unsupported Rate (UR) in our results, as the Dissonant Rate can be trivially calculated from the other two (1 – CR – UR).

E. Results

Our evaluation focuses on three key areas: establishing the Correctness of our system's performance metrics relative to generative coherence, analyzing the correlation between system success and music quality, and demonstrating the general feasibility of real-time accompaniment across various constraints.

Correctness of The Proposed Model Through extensive experimentation, we conclusively validate the correctness of the solution space derived from our mathematical model, which predicts feasible real-time configurations. To demonstrate this accuracy, we utilize the Weighted Interaction Success Rate (ISR$_w$), which quantifies the system's reliability and successful delivery rate, as the core experimental measure of system viability. The experimental data provides robust confirmation for our theoretical predictions. For example (see Fig. 4), under the local setting and BPM = 120, multiple configurations that fall within our mathematically defined solution space—such as (I=2, GL=5), (I=4, GL=9), (I=4, GL=15), and (I=7, GL=15)—all achieve universally high ISR$_w$ values of 0.9781, 0.9511, 0.9495, and 0.8628 respectively. This strong consistency, where configurations predicted as feasible by our model exhibit high real-time success (ISR$_w$), provides direct and empirical evidence supporting the fundamental correctness of our mathematical solution space. This outcome confirms that our theoretical framework effectively and accurately identifies the optimal settings for real-time system deployment.

Relations between System performance and Music Quality The experimental results establish a crucial synergistic relationship: successful system performance (ISR$_w$) acts as a strong, reliable proxy for high music quality. This is first evidenced by the validating relationship between ISR$_w$ and generative coherence (NLL). High ISR$_w$ correlates strongly with low NLL$_{\text{wavg}}$, confirming that reliable system delivery maintains the model's intrinsic statistical fluency. Conversely, the data reveals that NLL$_{\text{wavg}}$ is shown to be invalidated when ISR$_w$ is low. For example, in Table IIIb, a scenario with the lowest NLL$_{\text{wavg}}$ ($\mathbf{1.079}$) registers a low ISR$_w$ ($\mathbf{0.0373}$). We believe the primary cause of this confusing phenomenon is that a system failure state yields sparse sequences containing a high proportion of 'empty' tokens. When these sparse sequences are measured, the NLL is artificially low, failing to reliably indicate the quality of the few generated notes. Therefore, the invalidation of NLL$_{\text{wavg}}$ when ISR$_w$ is low further underscores that system success is the prerequisite for generative metrics to accurately reflect quality. NLL$_{\text{wavg}}$ is a reliable indicator of generative coherence only above a high ISR$_w$ threshold. Beyond generative coherence, the data confirms a strong positive correlation between system success (ISR$_w$) and most objective musical quality metrics. As ISR$_w$ improves (increases), the music quality significantly improves across multiple domains: the JSD(Onset) (rhythmic timing) decreases, the Consonance Ratio (CR) increases, and the Undergeneration Rate (UR) decreases. This strong, multi-metric correlation validates ISR$_w$ as a robust proxy for real-time music performance, confirming that successful, low-latency delivery of notes inherently leads to superior musical results. For instance, the best local setting (Table IIIa, Row (1, 3)) achieves an ISR_w of $\mathbf{0.9951}$ alongside high CR ($\mathbf{0.8207}$), excellent timing metrics, and an acceptable FMD, clearly demonstrating this synergy. The exception, JSD(Pitch), shows minimal correlation as it reflects the model's static harmonic vocabulary rather than the dynamic success or timing measured by ISR_w.

Feasibility in General The experiments across the three tables (Local, Server, Remote) confirm the general feasibility of establishing various successful real-time accompaniment systems. While server and remote latencies significantly reduce the system's success rate, the system consistently provides a comprehensive solution space of feasible $(\mathrm{I}, \mathrm{GL})$ configurations. For instance, even under challenging remote conditions (Table IIIc), we identified settings that maintain a functional ISR_w of $\mathbf{0.8785}$, proving the robustness of our architecture against diverse network constraints. Crucially, the overall experimental trend highlights that the best performance is consistently achieved at the highest possible frequency (lowest Inference Interval I). This aligns with the fundamental requirement of real-time interaction: to provide immediate feedback and maximal synchronization with the performer. The optimal configurations, such as $(\mathbf{I} = \mathbf{1}, \mathbf{GL} = \mathbf{3})$ in the local setting, demonstrate that high-frequency streaming inference, when constraints allow, is the superior strategy for maintaining both system reliability and generative coherence. This finding validates a core premise of our work: engineers and users should prioritize minimizing the Inference Interval (I) within the limits of their deployment environment to effectively balance music quality (low FMD, high CR) against specific constraints and maximize real-time responsiveness.

VI. Conclusion

This work presents a case study that establishes useful principles for building responsive, time-critical interactive transformer systems. By exploring the use case in accompaniment generation, systematically analyzing latency, synchronization, and inference behavior under real-time constraints, we provide a practical rule book for developers and researchers seeking to bridge the gap between algorithmic capability and interactive responsiveness. Our proposed StreamMUSE, to our best knowledge, is the first real-time accompaniment system that poses no musical constraints to user-input melodies. By focusing on real-time and cross-hardware adaptivity, our findings highlight key trade-offs between model complexity, system design, and perceptual thresholds, offering concrete guidance for future work on AI systems that must think, react, and create in real time.

References

[1] B. Fu, D. Yu, M. Liao, C. Li, Y. Chen, K. Fan, and X. Shi, "Efficient and adaptive simultaneous speech translation with fully unidirectional architecture," 2025. [Online]. Available: https://arxiv.org/abs/2504.11809

[2] J. Kasai, N. Pappas, H. Peng, J. Cross, and N. A. Smith, "Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation," 2021. [Online]. Available: https://arxiv.org/abs/2006.10369

[3] Y. Ren, C. Hu, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, "Fastspeech 2: Fast and high-quality end-to-end text to speech," 2022. [Online]. Available: https://arxiv.org/abs/2006.04558

[4] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," 2020. [Online]. Available: https://arxiv.org/abs/2005.00341

[5] R. Thoppilan, D. D. Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, Y. Li, H. Lee, H. S. Zheng, A. Ghafouri, M. Menegali, Y. Huang, M. Krikun, D. Lepikhin, J. Qin, D. Chen, Y. Xu, Z. Chen, A. Roberts, M. Bosma, V. Zhao, Y. Zhou, C.-C. Chang, I. Krivokon, W. Rusch, M. Pickett, P. Srinivasan, L. Man, K. Meier-Hellstern, M. R. Morris, T. Doshi, R. D. Santos, T. Duke, J. Soraker, B. Zevenbergen, V. Prabhakaran, M. Diaz, B. Hutchinson, K. Olson, A. Molina, E. Hoffman-John, J. Lee, L. Aroyo, R. Rajakumar, A. Butryna, M. Lamm, V. Kuzmina, J. Fenton, A. Cohen, R. Bernstein, R. Kurzweil, B. Aguera-Arcas, C. Cui, M. Croak, E. Chi, and Q. Le, "Lamda: Language models for dialog applications," 2022. [Online]. Available: https://arxiv.org/abs/2201.08239

[6] K. Shuster, J. Xu, M. Komeili, D. Ju, E. M. Smith, S. Roller, M. Ung, M. Chen, K. Arora, J. Lane, M. Behrooz, W. Ngan, S. Poff, N. Goyal, A. Szlam, Y.-L. Boureau, M. Kambadur, and J. Weston, "Blenderbot 3: a deployed conversational agent that continually learns to responsibly engage," 2022. [Online]. Available: https://arxiv.org/abs/2208.03188

[7] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[8] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," 2023. [Online]. Available: https://arxiv.org/abs/2309.06180

[9] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng, "Sglang: Efficient execution of structured language model programs," 2024. [Online]. Available: https://arxiv.org/abs/2312.07104

[10] Sumaiya, R. Jafarpourmarzouni, S. Lu, and Z. Dong, "Enhancing real-time inference performance for time-critical software-defined vehicles," in *2024 IEEE International Conference on Mobility, Operations, Services and Technologies (MOST)*, 2024, pp. 101–113.

[11] R. B. Dannenberg, "An on-line algorithm for real-time accompaniment," in *Proceedings of the International Computer Music Conference (ICMC)*, 1984, pp. 193–198.

[12] N. Orio, S. Lemouton, and D. Schwarz, "Score following: State of the art and new developments," in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, 2003, pp. 36–41.

[13] Y.-N. Hung and Y.-H. Yang, "Frame-level instrument recognition by timbre and pitch," in *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, 2018, pp. 135–142. [Online]. Available: https://arxiv.org/abs/1806.09587

[14] P. Esling, A. Chemla-Romeu-Santos, and A. Bitton, "Generative timbre spaces: Regularizing variational auto-encoders with perceptual metrics," in *Proceedings of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, 2018, pp. 150–157. [Online]. Available: https://arxiv.org/abs/1805.08501

[15] R. Rowe, *Interactive Music Systems: Machine Listening and Composing*. Cambridge, MA: MIT Press, 1993.

[16] F. Pachet, "The continuator: Musical interaction with style," *Journal of New Music Research*, vol. 32, no. 3, pp. 333–341, 2003.

[17] A. Cont, "A coupled duration-focused architecture for real-time music to score alignment," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 6, pp. 974–987, 2010.

[18] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer: Generating music with long-term structure," in *International Conference on Learning Representations (ICLR)*, 2019, arXiv:1809.04281. [Online]. Available: https://arxiv.org/abs/1809.04281

[19] H.-W. Dong, K. Chen, S. Dubnov, J. McAuley, and T. Berg-Kirkpatrick, "Multitrack music transformer," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023.

[20] J. Zhao and G. Xia, "Accomontage: Accompaniment arrangement via phrase selection and style transfer," 2021. [Online]. Available: https://arxiv.org/abs/2108.11213

[21] Z. Wang, L. Min, and G. Xia, "Whole-song hierarchical generation of symbolic music using cascaded diffusion models," 2024. [Online]. Available: https://arxiv.org/abs/2405.09901

[22] J. Thickstun, D. Hall, C. Donahue, and P. Liang, "Anticipatory music transformer," 2024. [Online]. Available: https://arxiv.org/abs/2306.08620

[23] N. Jiang, S. Jin, Z. Duan, and C. Zhang, "Rl-duet: Online music accompaniment generation using deep reinforcement learning," 2020. [Online]. Available: https://arxiv.org/abs/2002.03082

[24] C. Donahue, H. H. Mao, C.-Z. A. Huang, I. Simon, G. W. Cottrell, and D. Eck, "Bachduet: A benchmark dataset for interactive music generation," in *Proceedings of the NeurIPS 2022 Datasets and Benchmarks Track*, 2022. [Online]. Available: https://openreview.net/forum?id=Lt9x36MxdI5

[25] A. Scarlatos, Y. Wu, I. Simon, A. Roberts, T. Cooijmans, N. Jaques, C. Tarakajian, and C.-Z. A. Huang, "Realjam: Real-time human-ai music jamming with reinforcement learning-tuned transformers," 2025. [Online]. Available: https://arxiv.org/abs/2502.21267

[26] L. Blanchard, P. Naseck, S. Brade, K. Lecamwasam, J. Rudess, C.-Z. A. Huang, and J. Paradiso, "The jam_bot: A real-time system for collaborative free improvisation with music language models," in *Proceedings of the 26th International Society for Music Information Retrieval Conference (ISMIR)*, Daejeon, South Korea, 2025, mIT Media Lab and MIT Music Technology. [Online]. Available: https://arxiv.org/abs/2503.02165

[27] J. Jiang, D. Chin, L. Lin, X. Liu, and G. Xia, "Versatile symbolic music-for-music modeling via function alignment," 2025. [Online]. Available: https://arxiv.org/abs/2506.15548

[28] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[29] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference," *Proceedings of machine learning and systems*, vol. 5, pp. 606–624, 2023.

[30] Z. Wang*, K. Chen*, J. Jiang, Y. Zhang, M. Xu, S. Dai, G. Bin, and G. Xia, "Pop909: A pop-song dataset for music arrangement generation," in *Proceedings of 21st International Conference on Music Information Retrieval, ISMIR*, 2020.

[31] W.-Y. Hsiao, J.-Y. Liu, Y.-C. Yeh, and Y.-H. Yang, "Compound Word Transformer: Learning to compose full-song music over dynamic directed hypergraphs," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI*, 2021.

[32] J. Retkowski, J. Stepniak, and M. Modrzejewski, "Frechet music distance: A metric for generative symbolic music evaluation," 2025. [Online]. Available: https://arxiv.org/abs/2412.07948