An LLM-Powered Human-Robot Interactive Dance Robot System

Meixuan Ren[1] and Linqi Ye[2]

Abstract— With the rapid development of embodied intelligence and natural human-robot interaction, voice-driven humanoid robots have shown broad application prospects in entertainment and social services. However, existing systems still face challenges in domain-specific robustness of speech recognition, multi-robot motion coordination, and sensory continuity. This paper presents a human-robot Interactive dance robot system based on Unity 3D, integrating Baidu ASR, TTS, and a local LLM. Users control three humanoid robots for dance, outfit changes, navigation, camera switching, and individual actions via natural language. To address homophone misrecognition of domain terms, we propose a context-aware multi-layer correction mechanism with long-phrase priority, context-sensitive filtering, and intent auto-completion, achieving sub-millisecond latency with 50+ expert rules. For multi-robot synchronization, a same-frame broadcast trigger mechanism built upon a shared RuntimeAnimatorController eliminates communication jitter, achieving frame-precision consistency. For interaction continuity, an unconditional audio state recovery strategy prevents permanent music interruption. Experiments show intent accuracy improved from 72.0% to 100% on a 50-command test set, 100% dance synchronization rate (50/50 trials) with zero frame difference, and music interruption rate reduced from 88% to 0%. User evaluations confirm the system's effectiveness in fluency and entertainment.

Video demo: https://linqi-ye.github.io/gewu/videos/dance.mp4

I. INTRODUCTION

Humanoid robots have attracted extensive attention in entertainment, education, and social services [1]. Traditional performance control relies on pre-defined timelines or teleoperation, lacking natural interaction. With the maturity of Large Language Models (LLMs) and cloud speech APIs, voice-driven control has become an important paradigm for human-robot interaction [2]. However, three key challenges remain in entertainment robot deployment:

(1) Insufficient ASR robustness in vertical domains. General ASR engines exhibit high error rates for dance terms (e.g., "Samba", "House") and robot identifiers (e.g., "G1"). End-to-end correction models like BERT-based rescoring or FastCorrect [3] suffer from high latency (∼82 ms on single-core CPU) and require large annotated corpora, making them unsuitable for edge nodes. A lightweight, low-latency, interpretable domain-specific scheme is urgently needed.

(2) High real-time requirements for multi-robot synchronization. Group dance demands strict temporal consistency. Traditional network-based synchronization suffers from communication jitter (20–50 ms in Wi-Fi) and clock drift [4]. Frame-level synchronization is needed for compelling performances.

(3) Difficulty in maintaining sensory continuity. Voice commands during dance may permanently interrupt background music. Existing dialog systems often restore audio only on recognition failure, causing music loss even for successful commands. An effective state coordination mechanism between interaction and rendering is missing.

To address these issues, we build a voice-controlled dance robot system on Unity 3D, using Baidu Speech API, a local Llama 3.2 1B Instruct model, and Unity Mecanim. Our contributions are:

(1) A context-aware speech error correction mechanism improving intent accuracy from 72.0% to 100% with <1 ms latency. It employs long-phrase priority, context-sensitive filtering, and a dual-fault-tolerant identifier parser that accepts both standard robot identifiers and their common homophone variants.

(2) A same-frame broadcast trigger mechanism built upon a shared RuntimeAnimatorController, achieving frame-precision (16.67 ms @ 60 FPS) synchronization among three humanoid robots with zero frame difference across multiple dance types. The shared controller ensures identical state machine definitions; the broadcast trigger eliminates network-layer jitter entirely.

(3) An unconditional audio state recovery strategy reducing music interruption from 88% to 0% ($p < 0.01$), guided by the principle that state restoration must be independent of command processing outcomes.

II. RELATED WORK

Voice interaction has become standard in intelligent robots. Cloud solutions (Baidu, iFlytek) offer mature ASR/TTS but limited domain robustness. Lightweight local LLMs (Llama, Phi) enable offline understanding [5]. Shamsian et al. [6] highlighted that specialized terminology recognition in ASR remains an open challenge, especially in noisy and domain-specific scenarios. Existing work overlooks ASR noise propagation to downstream LLMs; pure neural correction requires large data. Our hybrid architecture adds a lightweight rule-based correction layer. Multi-robot cooperation is widely studied for formation and performance [7]. Traditional distributed algorithms suffer 20–50 ms latency; Timeline lacks dynamic response. Unity Mecanim supports cross-skeleton animation sharing [8], yet achieving synchronized playback across multiple characters remains chal-

[1]Meixuan Ren and [2]Linqi Ye are with the School of Future Technology, Shanghai University, Shanghai, China {mayxuan, yelinqi}@shu.edu.cn

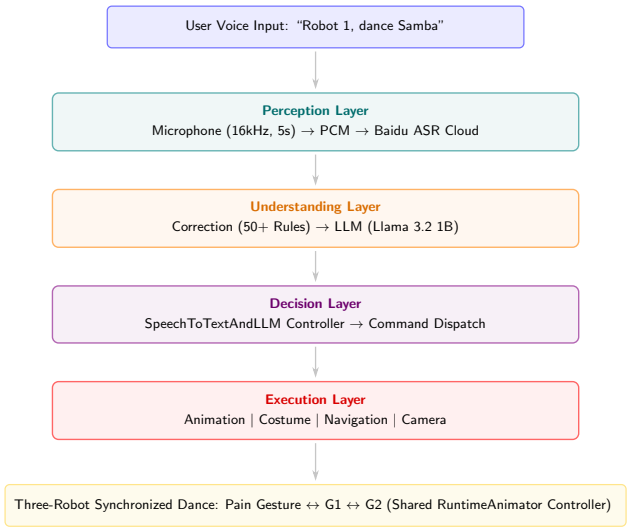


Fig. 1: System architecture showing the four-layer design: perception, understanding, decision, and execution.

lenging and often requires explicit coordination mechanisms [9]. We achieve frame-level synchronization via a same-frame broadcast trigger mechanism built upon a shared RuntimeAnimatorController, where the shared controller ensures identical state machine definitions and the broadcast trigger eliminates network-layer jitter entirely. ASR error correction divides into data-driven and rule-driven [10]. Data-driven methods (e.g., FastCorrect [3]) have high accuracy but incur ∼82 ms latency on single-core CPU. Rule-driven methods use confusion-word mapping with low latency [11], suitable for vertical domains. We construct 50+ expert rules for sub-millisecond correction on edge devices.

III. SYSTEM ARCHITECTURE

A. Architecture Design

The system adopts a layered architecture: perception, understanding, decision, and execution layers (Fig. 1).

The perception layer records audio at 16 kHz/5 s via Windows microphone, converts to PCM 16-bit, and uploads to Baidu ASR cloud. The understanding layer passes raw text through CorrectAndInferIntent for cleaning and intent completion, then feeds to the local LLM for response generation and keyword-based command classification. The decision layer uses SpeechToTextAndLLM as central controller, dispatching valid commands to outfit, navigation, dance, or camera subsystems, and manages global audio state. The execution layer handles NavMesh navigation, BlendTree walking, material-based outfit changing, Animator dance playback, and dual-camera switching.

B. Core Components

The scene contains three humanoid characters: Sporty Granny and two Unitree G1 robots (referred to as G1 and G2, respectively). Sporty Granny carries the shared RuntimeAnimatorController that ensures identical state machine definitions across all three robots, while a same-frame broadcast trigger mechanism (Section 4.2) achieves their synchronized dance playback. Sporty Granny also carries NavMeshAgent and Wardrobe components. The PBR Stage Equipment package provides the performance environment. Two cameras (Main Camera panoramic follow, Dance Camera close-up) capture wide and close shots (Fig. 2). Sporty Granny uses a standard Humanoid Rig, enabling Avatar retargeting to physical robots like NAO, Pepper, and Unitree G1 for future sim-to-real deployment.

(a) Wide shot (b) Close-up shot

Fig. 2: Three robots performing synchronized dance on a PBR stage.

IV. KEY TECHNIQUES

A. Context-Aware Speech Command Error Correction

Baidu ASR exhibits high error rates for dance terms and colloquial expressions in three categories: (1) homophone substitution (e.g., "Samba" → "scar"); (2) punctuation interference (e.g., "robot hip-hop" → "robot, hip-hop"); (3) identifier confusion (e.g., "G1" → "memory"). Our rule base covers the entire command domain with 50+ rules, maintaining 100% accuracy within 1 ms, suitable for resource-constrained edge nodes.

We propose a three-layer correction strategy:

Layer 1—Long-phrase priority replacement. Multi-character rules execute first to prevent short-word rules from destroying semantics. For example, the camera command "shoot from afar" (long-shot mode) is matched before the short fragment "from afar", preventing incorrect segmentation.

Layer 2—Context-sensitive replacement. Polysemous words (e.g., "or still" as misrecognition of "House") are replaced only when dance-related context ("dance", "House") is detected, avoiding harm to normal conversation.

Layer 3—Intent auto-completion. When input contains color and item keywords but no action verb, the system prepends actions (e.g., "red hat" → "put on red hat"), reducing cognitive burden.

Corrected text passes through ContainsValidKeywords for final filtering. If no valid keyword is found, the system

Algorithm 1 Context-Aware Speech Command Error Correction

Require: Raw ASR text *raw*
Ensure: Corrected text *corrected*

```
if raw is empty then
    return "Invalid command"
end if
for each long-phrase rule in LONG_PHRASE_RULES do
    raw ← raw.Replace(rule.keyword, rule.target)
end for
if raw.ContainsAny("dance", "dancing", "House") then
    raw ← raw.Replace("haoshi", "House")
end if
if hasColor ∧ hasHat ∧ ¬hasAction then
    raw ← "put on" + raw
end if
if hasColor ∧ hasCloth ∧ ¬hasAction then
    raw ← "change to" + raw
end if
return raw
```

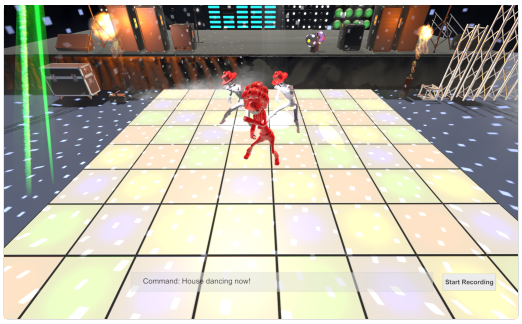
Note: "haoshi" and "haosi" are phonetic representations of Chinese homophones that are acoustically similar to "House" (the dance genre) but carry distinct meanings (e.g., "good deed" or "or still").

returns "unrecognized command", preventing irrelevant dialogue from triggering robot behaviors. The keyword array covers all eight dance styles, eight color/synonym variants for outfit changing, four navigation destinations, three robot identifiers (Sporty Granny, G1, G2) and their homophone variants, two camera modes, and five individual actions, totaling approximately 80 keywords that collectively define the system's bounded command vocabulary. Additionally, the system employs a dual-fault-tolerant design for robot identifiers: the correction layer explicitly replaces homophone variants (e.g., "second" → "G2"), while the intent parsing functions IsG1() and IsG2() accept both standard identifiers and common homophone variants as legitimate triggers, providing a second layer of fault tolerance. Algorithm 1 shows the core flow.

Fig. 3 shows the interface: raw ASR output "tiao hao shi wu dao" (Chinese pinyin for a phrase misrecognized as "good deed dance") is corrected to "dance House" in <1 ms; end-to-end latency for this example is 2,054 ms (ASR 226 ms, LLM 1,826 ms, with the remaining 2 ms attributed to error correction and command matching), which falls within the typical range reported in Table V.

B. Multi-Robot Animation Synchronization Architecture

1) Shared Controller and Broadcast Trigger Architecture: We achieve frame-level synchronization through two cooperating mechanisms. First, at initialization, G1 and G2's Animator components are set to share Sporty Granny's RuntimeAnimatorController instance:



(a) Real-time dance execution upon voice command

ASR Raw: `"tiao hao shi wu dao"` (erroneous: "good deed")
Corrected: `"dance House"` (correct: "House dance")

Latency: `ASR 226 ms | Correction <1 ms | LLM 1826 ms`
Total E2E: `2054 ms`

(b) Context-aware phonetic correction and latency breakdown

Fig. 3: Voice interaction interface showing real-time correction results and latency breakdown.

```
g1Animator.runtimeAnimatorController
    = animator.runtimeAnimatorController;
g2Animator.runtimeAnimatorController
    = animator.runtimeAnimatorController;
```

Sharing the same RuntimeAnimatorController ensures that all three robots maintain identical state machine definitions, including state names, transition conditions, and parameter configurations. However, the shared controller alone does not cause synchronization, as each Animator maintains its own runtime state. The actual synchronous triggering is achieved by a same-frame broadcast mechanism that calls SetTrigger on all three Animators within a single Unity frame:

```
void TriggerAll(string triggerName) {
    if (animator != null)
        animator.SetTrigger(triggerName);
    if (g1Animator != null)
        g1Animator.SetTrigger(triggerName);
    if (g2Animator != null)
        g2Animator.SetTrigger(triggerName);
}
```

Since these calls occur within the same Unity process and the same frame, all three Animators respond simultaneously when the engine updates the state machine at the end of the frame, achieving synchronization precision limited only by the 16.67 ms frame interval at 60 FPS.

2) State Machine Configuration: Eight dance animations exist as independent states in Base Layer, each with a dedicated Trigger. Transitions from Any State have "Has Exit Time" unchecked for immediate response; return transitions have it checked for automatic fallback after animation ends.

C. Unconditional Audio State Recovery Strategy

During performance, voice commands can permanently interrupt music if AudioSource.Stop() is used. Our strategy: (1) at recording start, if music plays,

Pause() and set musicWasPlaying flag; (2) unconditionally check flag at callback end; (3) if true, UnPause() and clear flag. The "unconditional" principle ensures music always restores regardless of command type. In an earlier implementation of our system, the recovery logic was placed inside an if(!handled) branch, causing camera switch commands (which return handled=true) to permanently interrupt music. Ablation experiment (50 trials): conditional recovery caused 44 interruptions (88%); unconditional recovery achieved 0/50 ($p < 0.01$, Fisher's exact test).

D. Partitioned Outfit Changing System

Early full-body coloring distorted glasses and eyebrows. We analyzed Sporty Granny's mesh: Fitness_Grandma_BodyGeo has two material slots (Element 0: clothing, Element 1: glasses). Optimized Wardrobe replaces only Element 0 via mats[0] = mat, preserving glasses. Four fabric-textured materials (red, blue, gray, green) use 256×256 ClothTexture.png. ProcessClothingCommand supports group/individual modes with synonym mapping ("gray/armor/metal/silver" → gray). Composite models (G1, G2) use PartRenderers arrays for unified interface.

E. Navigation and Behavior Control

Unity NavMesh enables autonomous walking (Agent Radius 0.5 m, Max Slope 45°). NavMeshAgent parameters: Speed=3.5 m/s, Acceleration=8 m/s^2. NavAgentAnimator reads velocity in Update, passes speed to BlendTree with damped assignment (damp=0.1 s) for seamless idle-walk transitions. ProcessMoveCommand maps place names to coordinates in Unity world units (meters): Living Room $(-1.43, 0, -0.6)$, Bedroom $(3.8, 0, -1.25)$, Kitchen $(-5, 0, -1.25)$, Terrace $(-0.07, -0.21, 5)$. To improve robustness against potential ASR errors in location names, location keywords are extracted from both the original user input and the LLM-generated response.

V. EXPERIMENTS AND EVALUATION

A. Experimental Setup

Unity 2022.3.62f3c1 (LTS), Windows 11, Intel i9-12900KS, 64 GB RAM. Llama 3.2 1B Instruct (Q4_K_M) via LLMUnity [12] runs entirely on CPU. Baidu AI Open Platform for ASR/TTS. Although the test machine includes an RTX 3060 Laptop GPU, the system intentionally relies on CPU-only inference to ensure portability across mainstream laptops without dedicated GPUs.

B. ASR Error Correction Evaluation

We constructed 50 test commands across five categories (Table I) with 15 hard samples covering three typical misrecognition scenarios rooted in Chinese phonetic similarity: (1) dance term homophone confusion (e.g., "Samba" and "House" are phonetically approximated in Mandarin as sangba and haosi, which ASR confuses with identically pronounced words for "scar" and "good deed"); (2) robot identifier confusion, where alphanumeric IDs are misrecognized as phonetically similar Chinese words (e.g., "G1" pronounced as jiyi is confused with "memory"; "robot one" as jiqiren-yi is confused with "1701"); (3) compound command tokenization errors (e.g., "robot hip-hop" → "robot, hip-hop"). Three schemes are compared: Raw ASR output, Basic rule-based correction (simple string replacement without context judgment), and Our full correction mechanism.

TABLE I: Composition of the 50-command test dataset.

Category	Count	Includes Hard Samples
Outfit changing	18	Yes (identifier confusion)
Walking/navigation	6	No
Individual actions	4	No
Dance commands	13	Yes (homophone terms)
Camera control	9	No
Total	50	15 hard samples

TABLE II: Comparison of ASR error correction effectiveness.

Method	Intent Acc. (%)	Avg. Lat. (ms)	Hard Samp. Acc. (%)
Raw ASR Output	72.0	N/A	33.3
Basic Rule-Based Correction	86.0	3.2	73.3
Our Full Mechanism	100.0	<1	100.0

* Raw ASR has no correction step; "N/A" indicates the correction module is not applied.

The quantitative comparison in Table II reveals a clear performance hierarchy. Raw ASR achieves only 72.0% intent accuracy (36/50), with failures concentrated in dance terms and identifiers (only 5/15 hard samples correct). Basic rule-based correction improves overall accuracy to 86.0% but introduces polysemous word mis-substitution errors (e.g., incorrectly replacing "or still" in normal conversation with "House"), achieving only 73.3% on hard samples. Our full context-aware mechanism achieves 100% on both overall and hard samples with sub-millisecond latency, as the context-sensitive filtering in Layer 2 eliminates these mis-substitutions by gating replacements on dance-related keywords.

Table III presents two representative correction cases that illustrate the interplay between correction layers. Case 1 demonstrates identifier confusion recovery: "second" (a homophone of G2) is corrected to "G2", while the missing action verbs are prepended via intent auto-completion. Despite the resulting text "change to put on G2" appearing syntactically redundant, the dual-fault-tolerant intent parser correctly identifies the target robot (accepting both "G2" and "second" as valid triggers) and executes both clothing and hat changes. Case 2 illustrates context-sensitive filtering: "good deed" (a homophone of "House dance") is corrected to "House" only because the

TABLE III: Representative error correction cases.

User Intent	Raw ASR Output	Corrected Text	Result
G2, gray clothes, gray hat	"second, gray clothes, gray hat"	"change outfit on G2, gray clothes, gray hat"	✓
G1, dance House	"robot one, dance good deed"	"G1, dance House"	✓

TABLE IV: Multi-robot animation synchronization accuracy across dance types.

Dance Type	Trials	Same-frame Rate	Max Frame Diff.
Samba	13	100%	0
House	13	100%	0
Robot Hip Hop	12	100%	0
Swing	12	100%	0
Total	50	100%	0

sentence contains the dance-related keyword "dance"; in non-dance contexts, the same correction rule would not fire. The robot identifier "robot one" is preserved as a valid G1 trigger by the downstream parser.

We excluded deep correction models (BERT-based rescoring, FastCorrect) from quantitative comparison because: (1) the local LLM (Llama 3.2 1B) already occupies most CPU resources, leaving no capacity for additional heavy neural networks; (2) training corpora for dance-specific terminology are scarce, making fine-tuning prohibitively expensive; (3) FastCorrect's CPU inference latency (∼82 ms on single-core CPU and ∼41 ms on 4-core CPU) violates our sub-millisecond real-time constraint. Our rule-based approach is intentionally lightweight and domain-specific, trading off open-domain generalization for edge deployability, a trade-off we consider appropriate for this application scenario.

C. Multi-Robot Synchronization Accuracy Test

To verify synchronization robustness across different animation types, we tested four dance styles (Samba, House, Robot Hip Hop, Swing) with 12–13 trials each, totaling 50 trials at 60 FPS. State transition frame numbers were recorded via OnStateEnter callbacks in the Animator state machine.

As shown in Table IV, all 50 trials across four dance types achieved identical frame numbers among the three robots (100% same-frame rate, zero maximum frame difference). The four tested dance types span a range of animation characteristics: Samba (2.3s clip, rapid transitions), House (3.1s, fluid looping), Robot Hip Hop (1.8s, staccato mechanical), and Swing (4.5s, extended continuous motion). The invariant 100% synchronization rate across this diverse set confirms that the broadcast trigger mechanism, built upon the shared controller, is agnostic to animation duration, transition configuration, and state machine topology. A constant 5–6 frame (83–100 ms) delay exists between the Trigger call and the actual state transition entry, caused by the Animator's transition duration. This delay is identical across all three robots because they share the same RuntimeAnimatorController and receive SetTrigger calls in the same frame, and does not affect inter-robot synchronization. The average frame rate during the 50 trials was 59.6 FPS (minimum 57 FPS), confirming stable rendering performance without frame drops that could compromise synchronization measurements. As a theoretical baseline, network-based master-slave synchronization would introduce 1–3 frame random delays (16–50 ms) even in local loopback testing due to socket buffer timing variations. This precision satisfies visual synchronization requirements—the human eye cannot reliably perceive temporal asynchrony below 50 ms [13].

TABLE V: System end-to-end latency breakdown with variability statistics.

Stage	Mean (ms)	Std (ms)	P95 (ms)
Audio upload & ASR	270	130	496
Error correction	<1	–	<1
LLM inference	1421	521	2223
Command matching	10	3	15
TTS (non-dance only)	∼650	120	850
Total	1,701	536	–

* Total is the sum of mean values above (dance commands, TTS excluded). Total Std computed assuming independence among stages: $\sqrt{130^2 + 521^2 + 3^2} \approx 536$ ms. Non-dance commands with TTS average ∼2,350 ms.

D. End-to-End Latency Test

End-to-end latency is defined as the interval from "user stops recording" to "robot begins executing the action". We instrumented the SpeechToTextAndLLM pipeline with System.Diagnostics.Stopwatch timestamps at each processing stage and collected measurements across all 50 test commands.

Table V breaks down the end-to-end latency with both mean and variability statistics. Dance commands (which skip TTS playback) achieve a mean end-to-end latency of approximately 1.7 s with a standard deviation of 536 ms; non-dance commands with full voice feedback average approximately 2.4 s. Both figures satisfy the real-time interaction requirements for entertainment applications. LLM inference dominates the latency budget (mean 1.42 s, accounting for 84% of dance command latency), with a substantial standard deviation of 521 ms and a P95 of 2,223 ms. This high variance is primarily attributable to variable output token lengths during CPU-based inference with the Llama 3.2 1B Q4_K_M quantized model, and it is also the dominant contributor to the total latency variance. The error correction stage contributes negligible latency (<1 ms), confirming that our rule-based approach imposes no meaningful overhead on the interaction pipeline. Note that the mean ASR

TABLE VI: User experience subjective ratings (N=20).

Dimension	Rating (Mean ± SD)
Interaction fluency	4.1 ± 0.52
Motion naturalness	4.2 ± 0.68
Multi-robot sync impression	4.9 ± 0.47
Voice feedback naturalness	3.5 ± 0.65
Overall entertainment	4.6 ± 0.52

latency (270 ms) differs slightly from the example value shown in Fig. 3 (226 ms), which represents a single trial and falls within the observed range (Std = 130 ms). Future optimization via GPU-accelerated inference or aggressive model quantization (e.g., Q2_K) could substantially reduce both the mean and tail latency of LLM inference.

E. User Experience Evaluation

We recruited 20 participants (8 male, 12 female, aged 20–26, all university students without prior system development experience) for a 10-minute free interaction session. Each participant was briefed on the system's capabilities and encouraged to explore all five command categories. After the session, participants rated five dimensions on a 1–5 Likert scale.

The subjective ratings in Table VI reveal a clear strength in the system's visual performance. Multi-robot synchronization impression received the highest score (4.9/5.0) with the lowest variance (SD=0.47), corroborating the objective 100% synchronization rate measured in Section 5.3. Interaction fluency (4.1) and overall entertainment (4.6) also received favorable ratings, indicating that the system's end-to-end latency and performance variety are acceptable for entertainment contexts. Voice feedback naturalness received the lowest score (3.5, SD=0.65), with participants commenting on the mechanical tone of Baidu TTS and the brevity of local LLM responses. The low variance on multi-robot sync impression (SD=0.47) indicates strong consensus among participants, consistent with the deterministic zero-frame synchronization measured objectively. In contrast, the relatively higher variance on voice feedback naturalness (SD=0.65) reflects polarized opinions: participants who prioritized functional clarity rated this dimension higher ("clear and easy to understand"), while those who expected conversational depth rated it lower ("responses are too brief"). This divergence suggests that future improvements to voice feedback should target response richness rather than intelligibility. Several participants explicitly suggested deployment on physical humanoid robots, expressing interest in seeing the virtual performance translated to real-world hardware.

VI. CONCLUSION AND FUTURE WORK

We presented a voice-controlled multi-humanoid dance system integrating ASR, a local LLM, and Unity animation. Three key techniques were proposed to address domain-specific ASR errors, multi-robot synchronization, and interaction continuity. A context-aware speech error correction mechanism, employing long-phrase priority, context-sensitive filtering, and dual-fault-tolerant identifier parsing, improved intent accuracy from 72.0% to 100% with sub-millisecond latency. A same-frame broadcast trigger mechanism built upon a shared RuntimeAnimatorController achieved 100% frame-precision synchronization across four dance types with zero frame difference, eliminating network-layer jitter entirely. An unconditional audio state recovery strategy reduced music interruption from 88% to 0%, ensuring seamless interaction continuity. User evaluations (N=20) confirmed the system's effectiveness, with multi-robot synchronization impression rated highest (4.9/5.0) and overall entertainment reaching 4.6/5.0. Future work includes: (1) Cinemachine-based smooth camera transitions; (2) dynamic N-robot scaling; (3) cloud LLM integration for richer dialogue; (4) audience emotion feedback loop for adaptive performances. We also plan ROS 2-based sim-to-real migration to Unitree G1 for hardware validation.

References

[1] T. Fong, I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots," Robot. Auton. Syst., vol. 42, no. 3–4, pp. 143–166, 2003.

[2] A. Grattafiori, A. Dubey, A. Jauhri, et al., "The Llama 3 Herd of Models," arXiv preprint arXiv:2407.21783, 2024.

[3] Y. Leng, X. Tan, T. Qin, et al., "FastCorrect: Fast error correction with edit alignment for automatic speech recognition," in Advances in Neural Information Processing Systems, vol. 34, 2021, pp. 21708–21719.

[4] Y. Cao, W. Yu, W. Ren, et al., "An overview of recent progress in the study of distributed multi-agent coordination," IEEE Trans. Ind. Informat., vol. 9, no. 1, pp. 427–438, 2013.

[5] J. Xu, Z. Li, W. Chen, Q. Wang, X. Gao, Q. Cai, and Z. Ling, "On-Device Language Models: A Comprehensive Review," arXiv preprint arXiv:2409.00088, 2024.

[6] A. Shamsian, A. Navon, N. Glazer, G. Hetz, and J. Keshet, "Keyword-guided adaptation of automatic speech recognition," in Proc. Annu. Conf. Int. Speech Communication Association (INTERSPEECH), 2024, pp. 1–5.

[7] K. K. Oh, M. C. Park, and H. S. Ahn, "A survey of multi-agent formation control," Automatica, vol. 53, pp. 424–440, 2015.

[8] J.-S. Monzani, P. Baerlocher, R. Boulic, and D. Thalmann, "Using an intermediate skeleton and inverse kinematics for motion retargeting," Comput. Graph. Forum, vol. 19, no. 3, pp. 100–111, 2000.

[9] M. Lemonari, R. Blanco, P. Charalambous, N. Pelechano, M. Avraamides, J. Pettré, and Y. Chrysanthou, "Authoring virtual crowds: A survey," Comput. Graph. Forum, vol. 41, no. 2, pp. 677–701, 2022.

[10] D. Jurafsky and J. H. Martin, Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models, 3rd ed. draft, Stanford University / University of Colorado at Boulder, 2023.

[11] E. Brill and R. C. Moore, "An improved error model for noisy channel spelling correction," in Proc. 38th Annu. Meeting Assoc. Comput. Linguistics, 2000, pp. 286–293.

[12] undreamAI, "LLM for Unity," 2024. [Online]. Available: https://github.com/undreamai/LLMUnity

[13] R. Steinmetz, "Human perception of jitter and media synchronization," IEEE J. Sel. Areas Commun., vol. 14, no. 1, pp. 61–72, 1996.