LLMs can read music, but struggle to hear it. An evaluation of core music perception tasks

Brandon James Carone BCARONE@NYU.EDU
Department of Psychology, Music and Audio Research Laboratory (MARL), Center for Language, Music, and Emotion (CLaME), New York University

Iran R. Roman I.ROMAN@QMUL.AC.UK
School of Electronic Engineering and Computer Science, Queen Mary University of London

Pablo Ripollés PRIPOLLES@NYU.EDU
Department of Psychology, Music and Audio Research Laboratory (MARL), Center for Language, Music, and Emotion (CLaME), New York University

Abstract

Multimodal Large Language Models (MLLMs) claim "musical understanding," yet most evaluations conflate listening with score reading. We benchmark three SOTA LLMs (Gemini 2.5 Pro, Gemini 2.5 Flash, and Qwen2.5-Omni) across three core music skills: Syncopation Scoring (rhythm perception), Transposition Detection (melody perception), and Chord Quality Identification (harmony perception). Moreover, we separate three sources of variability: (i) perceptual limitations (by contrasting audio recordings vs. symbolic MIDI inputs), (ii) exposure to prior examples (zero- vs. few-shot manipulations), and (iii) reasoning strategies (Standalone, Chain of Thought, LogicLM). For the latter we adapt LogicLM, a framework combining LLMs with symbolic solvers to perform structured reasoning. In LogicLM, LLMs act as perceptual formulators, generating strict, machine-checkable schemas (onset grids, interval sequences) that deterministic solvers execute with self-refinement. Our results reveal a clear perceptual gap: models perform near ceiling on MIDI but show substantial accuracy drops on audio. Reasoning and few-shot prompting offer minimal gains. This is expected for MIDI, where performance reaches saturation, but more surprising for audio, where LogicLM, despite near-perfect MIDI accuracy, remains notably brittle. Among models, Gemini Pro achieves the highest performance across most conditions. Transposition yields the highest accuracies across models, while Chord Identification scores slightly below Syncopation. Overall, current systems reason well over symbols (MIDI) but do not yet "listen" reliably from audio, with reasoning strategies having little impact over accuracy. Our method and dataset make the perception–reasoning boundary explicit and offer actionable guidance for building robust, audio music systems.

Keywords: Audio Large Language Models, Multimodal Large Language Models, Music Understanding, Benchmarking and Evaluation, Schema-Guided Reasoning, LogicLM

1. Introduction

Recent advances in foundation models have extended their reach beyond text to multimodal architectures that process audio, vision, and language in a unified framework. Models such as Alibaba's Qwen2.5-Omni, trained on a range of audio tasks (Xu et al., 2025), and Google's Gemini 2.5 family, which incorporates advanced multimodal integration for real-time interactions (Comanici et al., 2025), exemplify this new generation. One problem with many multimodal LLMs is that they boast "generic hearing abilities" and "music understanding", yet struggle with tasks as simple as recognizing a well-known tune when transposed to a different key (i.e., singing the same song at a higher or lower pitch) or played on another

instrument. For example, after being fed a simple piano rendition of "Happy Birthday" and told that the melody represents that tune regardless of which key it is played in, they fail to recognize it when played in a different key or on another instrument (in our own pilot tests, Gemini 2.5 guesses that the transposed version of "Happy Birthday" played at the same tempo is "Twinkle Twinkle Little Star" and Qwen2.5-Omni responded with "Lose Yourself" by Eminem). On the other hand, most people with Western enculturation recognize "Happy Birthday" across keys, instruments, and language (Halpern and Bartlett, 2010; Margulis, 2013). Often these models are evaluated on benchmarks (e.g., AIR-Bench; (Yang et al., 2024)) that primarily focus on tasks such as speech recognition, audio classification, or music tagging/captioning using publicly available datasets (e.g., MusicCaps, AudioSet, NSynth, MagnaTagATune; (Chu et al., 2023, 2024; Kong et al., 2024; Ghosh et al., 2025b; Tang et al., 2024; Liu et al., 2024)). Among the state-of-the-art audio benchmarks are MMAR (Ma et al., 2025b), MMAU (Sakshi et al., 2025), and MMAU-Pro (Kumar et al., 2025), CMI-Bench (Ma et al., 2025a), RUListening (Zang et al., 2025), and FUTGA-MIR (Wu et al., 2025), which broaden coverage across speech, sound, and music, emphasizing multi-step reasoning. These benchmarks add realism via in-the-wild audio, long-form and multi-audio settings, spatial understanding, and expert-crafted Question-Answer pairs. In the context of music, MMAU-Pro even asks open-ended questions regarding musical themes, a song's mix, and what might differentiate one song from another. However, no existing benchmarks strategically ask questions about music that test whether the model can "listen to" the specific relations that constitute musical structure, or do so robustly enough to form a symbolic representation for a piece of music, like how a musician might transcribe a song.

While these datasets may be effective in assessing isolated aspects of music understanding (e.g., genre and instrument identification), it is still unknown whether they fully capture the nuanced, hierarchical nature of music perception in human listeners. For example, Audio LLMs may learn to associate the spectral characteristics of a trumpet's timbre with the label "trumpet," or tie fast tempos, loud drums, and distorted guitars with the label "rock music," simply by maximizing the likelihood of these co-occurrences in the training data. However, this focus on surface statistics is less suited for tasks demanding abstract relational understanding, such as recognizing a melody when its absolute pitches are changed (key invariance) or identifying the harmonic function of a chord within a progression. These abilities require understanding relationships between elements (e.g., relative pitch intervals, harmonic contexts) rather than just recognizing the elements themselves.

In the current study, we focus on three fundamental components of musical hierarchy: rhythm, melody, and harmony. Syncopation captures the perception of rhythmic "surprise" or emphasis in unexpected places (Large et al., 2015, 2023). Evaluating a model's capacity to detect syncopation provides a measure of its sensitivity to temporal predictability and metric displacement in rhythm perception. Testing melody recognition across transpositions (Dowling and Fujitani, 1971; Dowling, 1978; Deutsch, 1969) assesses the model's ability to recognize melodies despite shifting the key, or rather, starting the same melody on a different pitch or note that is higher or lower and maintaining the rhythmic and intervallic changes. By testing this, we can evaluate whether the model exhibits similar perceptual invariance to humans. Identifying chord quality (i.e., major, minor, dominant, diminished) requires a model to recognize harmonic structures based on the relative intervals above the root note of the chord, rather than absolute pitch alone. By testing this, we can evaluate whether

the model demonstrates an ability to track intervals above a given note, and characterize the joint quality of the pitches forming those intervals as a whole.

Evaluating these abilities in multimodal LLMs presents a methodological challenge: how can we pinpoint where the bottleneck in musical abilities lies? To answer this question, we have developed an experimental design that jointly assesses perceptual, learning, and reasoning factors. First, we aim to test the perceptual limitations of multimodal LLMs by comparing their performance on audio recordings, which require genuine listening, versus symbolic MIDI inputs, which rely on structured representations. Second, we aim to assess how in-context learning (few-shot vs. zero-shot) influences accuracy, revealing whether brief exposure to examples improves music perception, or if performance remains limited by underlying perceptual constraints. And third, when a model produces a correct answer, it is often unclear whether this reflects genuine perceptual analysis (i.e., reasoning) or reliance on superficial cues. This echoes the problem of "unfaithful reasoning" in logical domains, where models may generate plausible chains of thought that do not underlie their actual decision process (Pan et al., 2023). Even Chain-of-Thought prompting (a technique where an LLM generates intermediate reasoning steps to improve the accuracy of its answer), does not necessarily guarantee alignment with the perceptual computations themselves. To address this, we adapt LogicLM (Pan et al., 2023), a neuro-symbolic prompting framework in which an LLM emits a strict, machine-checkable schema that a deterministic solver executes, aided by a self-refinement loop, so that the final answer is grounded in verifiable computation rather than free-form text or guessing. In our adaptation, the model serves as a Perceptual Formulator, tasked with converting continuous audio into a structured symbolic schema (e.g., note sequences, rhythmic onsets, pitch-class sets). These are then evaluated by deterministic solvers, ensuring that the final decision is grounded in the schema rather than in opaque model heuristics. By systematically comparing model performance on raw audio versus symbolic MIDI inputs, and by contrasting LogicLM prompting with standalone and Chain-of-Thought strategies, we provide a detailed assessment of where current models succeed, where they fail, and what this reveals about the limits of machine music perception.

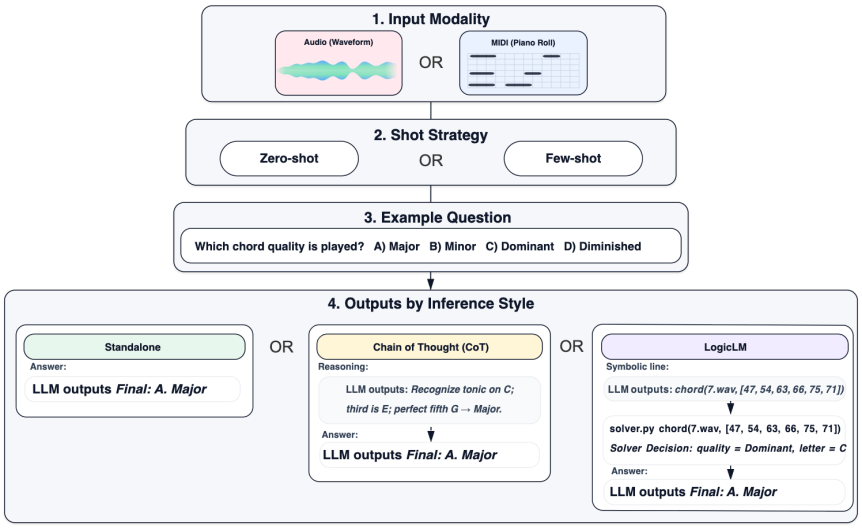

Figure 1: Diagram of the experimental design carried out with each model and task.

2. Methods

2.1. Stimuli Creation

Stimuli were recorded by a real human musician, and are originally from The MUSE Benchmark (Carone et al., 2025a,b). See Supplementary Materials S1 for details.

2.2. Tasks

Syncopation Scoring. In this task, the models were presented with 20 short rhythmic excerpts (8 secs) performed at 120 BPM on a drum set, consisting of kick, snare, and hi-hat. The hi-hat maintained a constant stream of eighth notes, while the kick and snare patterns varied in their placement across on-beats (those falling on the quarter notes of each bar) and off-beats (those falling on the 8th notes in between each quarter note). The models' task was to rate the degree of syncopation by counting the number of kick and snare events that occurred on off-beats, and then mapping this total to a categorical Syncopation Score (i.e., 0, 2, 4, 6, or 8). Stimuli were systematically constructed to span a wide range of syncopation levels, in accordance with the methods of (Large et al., 2015, 2023).

Transposition Detection. In this task, the models were presented with 20 pairs of musical excerpts (mean duration $\approx$ 9 secs) where the first excerpt presented is the anchor, and the second (i.e., target) is either the same melody transposed to a different key, or a different melody. After "listening", the models must decide whether the two audio clips represent the same melody or not. 10 of the trials were matches, and the other 10 were not matches. Stimuli were short excerpts played on an electric guitar or a piano, and were varied across tempo, key, meter, and melody length.

Chord Quality Identification. In this task, the models were presented with 44 short musical excerpts (9 secs) recorded at 120 BPM and consisting of a single chord played first as a block and then as an arpeggiation, where each of the individual notes of the chord were played individually from lowest to highest. Each chord was in root position, where the lowest note is the root of the chord, to remove ambiguity about inversion, and all stimuli were generated on piano to ensure consistent timbre across all trials. The models' task was to classify the chord into one of four quality categories: Major (Root + major 3rd + perfect 5th), Minor (Root + minor 3rd + perfect 5th), Dominant (Root + major 3rd + perfect 5th + minor 7th), or Diminished (Root + minor 3rd + diminished 5th).

2.3. Implementation

We developed a set of custom inference scripts that closely followed the framework proposed in the original LogicLM study (Pan et al., 2023), which compared three prompting strategies: Standalone, Chain-of-Thought (CoT), and LogicLM. In adapting this design to the domain of music perception, we ensured that: (i) each trial was independent of the others, (ii) prompts were standardized across prompting strategies, and (iii) results could be evaluated in a reproducible way. Each trial began with a fresh chat session, meaning that no conversational history was ever preserved across trials or across tasks. All prompting strategies included the same set of task-specific system instructions, which defined the rules of the task and the required output format. In constructing our scripts, we wanted to

separate three key sources of variability: (i) perceptual limitations (by contrasting audio vs. symbolic inputs), and (ii) learning by exposure to prior examples (zero- vs. few-shot manipulations), and (iii) reasoning strategies (standalone vs. CoT vs. LogicLM). The differences between the different runs are outlined below:

Per-task modularity. Each task was implemented in a separate script. This ensured that stimuli, examples, and outputs were isolated per task and that no prompt information leaked across prompting strategies. Each condition was tested in the same structure, allowing direct comparisons across tasks.

Audio vs. MIDI data. We ran a symbolic-input control by replacing audio with MIDI notation for the same items. Prompts simply swap "you will hear..." for "you will be given MIDI data...", and the model is asked to generate the same schema as in the audio runs. All stimuli were rerecorded on a MIDI keyboard and then translated to .txt files using a custom script and the python package `mido`. This isolates the effect of perceptual transcription from symbolic reasoning.

Zero-shot vs. Few-shot. In zero-shot prompting strategies, models received only the system instructions and the trial stimuli. In few-shot prompting strategies, we included a small number of worked examples in the trial history (two for syncopation scoring and transposition detection; four, one per quality category, for chord quality detection), each paired with the correct solution. These examples were presented only for that trial and were excluded from the evaluation set. This separation allowed us to test whether models could solve tasks based on their intrinsic knowledge or whether they benefited from in-context learning from demonstrations.

Standalone, CoT, and LogicLM. In the standalone condition, models were asked to provide only the final categorical response (e.g., "Yes, these are the same melody.", "C. Dominant"). In the CoT condition, they were encouraged to produce short intermediate reasoning before giving a final answer on a separate line. In the LogicLM condition, the model was required to output a structured symbolic transcription (e.g., a list of note intervals or a grid of rhythmic onsets), which was then parsed by a deterministic solver (solver.py; see Supplementary Materials S3). When schema violations occurred (for example, malformed syntax or an out-of-range onset), we implemented a self-refinement loop in which the model was asked to correct its own output under strict constraints. This mirrors the iterative repair process described in (Pan et al., 2023). System instructions for each task and condition can be found in Supplementary Materials S2.

All responses from the LLMs were parsed with regular expressions to extract the final line (e.g., "Final Answer: B" or "Yes, these are the same melody."). For LogicLM, the symbolic output was passed to the solver, and solver decisions were used to score the trial. All trials across runs were randomized and logged to a dedicated file that included the model configuration, trial IDs, raw outputs, parsed responses, and evaluation results.

2.4. Models and inference environment

We tested Gemini 2.5 Pro, Gemini 2.5 Flash, and Qwen2.5-Omni. Gemini runs used the google.genai SDK, whereas Qwen runs mirrored the same pipeline on the NYU HPC

(SLURM), with provider-specific chat/message shims but the same prompts, decoding settings, seeding, and evaluation. All Qwen2.5-Omni experiments were run on the NYU Greene HPC cluster using SLURM. To run the scripts, we used 2 NVIDIA H100 GPUs, 64 GB of system memory, and 8 CPU cores to run all of the scripts.

2.5. Statistical Analyses

Decision Accuracy. Analyses were conducted in `Python` (`pandas` for data handling, `numpy` for numerical operations, and `matplotlib` for visualization). The unit of analysis was a run (one log file). For each run, we computed an accuracy score:

$$\text{Accuracy } \% = 100 \times \frac{\text{Correct}}{\text{EffectiveTotal}},$$

where *Correct* denotes the number of correctly scored trials and *EffectiveTotal* is the total number of trials after subtracting token-limited events (i.e., truncated or empty model outputs; 0.01% of all trials). Each run therefore contributed a single accuracy observation, annotated with metadata fields: *Task, Model, Modality, Condition*, and *Shot*.

Exploratory statistical analyses. After inspecting the decision accuracy results (see Table 1), we found that reasoning style (Standalone, CoT, or LogicLM) and shot setting (few-shot vs. zero-shot) had minimal effects, whereas accuracy differences were primarily driven by modality (audio vs. MIDI). Consequently, we conducted post hoc statistical analyses to examine LLM performance as a function of modality. We used generalized linear mixed modeling (GLMM) in R (version 4.4.2) and RStudio (2024.09.1) with the `lme4` package. Each response produced by the models for every stimulus served as an observation in the GLMM dataset. After removing the token-limited trials, separate GLMMs were fit for each task (Syncopation, Chord Identification, and Transposition), predicting whether each response was correct (1) or incorrect (0) at the trial level. Each model included fixed factors for Model (Gemini Flash, Gemini Pro, Qwen 2.5 Omni) and Modality (Audio, MIDI), as well as their interaction, with a random intercept for Stimulus: [Correct $\sim$ Model * Modality + (1 | Stimulus)]. The effects of the different predictors and interactions were evaluated using Type III Wald chi-square tests via the `car` package, and significant interactions were further explored using `emmeans`.

Quality of LogicLM Inputs x Accuracy of LLM Responses. Because the models seemed to fail at perception from the waveforms and not from the symbolic reasoning itself, we visualize, across task and modalities, how the quality of the symbolic inputs produced under LogicLM (x-axis) relates to the final, task-level decision accuracy of the LLM (y-axis). If perception is the bottleneck, Audio points should sit low on the x-axis (poor input quality), and low on the y-axis (poor decision quality). With clean symbolic inputs (MIDI), however, both axes should approach ceiling. To quantify the **LogicLM input quality** for each trial, we compute the F1 (a measure that combines precision and recall into a single metric) of the LLM predicted MIDI content (i.e., the symbolic representation generated by the LLM) vs. ground truth (annotations of the stimuli provided by a human expert). Let TP, FP, FN be true/false positives/negatives for the relevant set comparison. We compute

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}, \qquad \text{F1} = \frac{2\,\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (1)$$

The x–axis is this per–trial F1 on (i) onset sets for Syncopation (masked to on– or off–beats), (ii) absolute pitch–sets for Chord Quality ID, and (iii) pitch–class content for Transposition. The y–axis is the **decision accuracy** for the same trials (correct/incorrect).

Trials are aggregated *across models and shot settings* to produce model–agnostic summaries at the level of **Task label** × **Modality**:

- Syncopation: *On–beat* and *Off–beat* × {Audio, MIDI}
- Transposition: *Yes / No* × {Audio, MIDI}
- Chord Quality ID: *Major / Minor / Dominant / Diminished* × {Audio, MIDI}

Within each Task label × Modality grouping, we perform micro–averaging by summing TP, FP, and FN over all contributing trials (pooled over models and shots) and then computing precision, recall (i.e., sensitivity), and F1 from those totals, thereby weighting each trial equally and yielding an overall estimate that is robust to differing numbers of runs/models and label imbalance. Note that F1 ranges from 0 to 1, where 1 indicates perfect precision and perfect recall. Accuracy (whether the question was answered correctly) is computed as the number of correct responses divided by the total number of trials in that grouping.

For the plot, Syncopation items are partitioned into on-beat and off-beat subsets. For each level and modality, we compute F1 separately, while the single binary decision-accuracy score (correct/incorrect) for that trial is applied to both its on-beat and off-beat data points. This allows us to compare the models' beat tracking abilities (on-beats) with its ability to actually detect moments of syncopation (off-beats). For Transposition, the script extracts the predicted pitches from both the anchor and target stimuli, and computes the F1 against ground truth. For Chord Quality ID, the script takes the models' chord prediction and compares the predicted absolute note set to ground truth to obtain F1.

3. Results

Overall performance Table 1 summarizes accuracy across tasks, models, modalities, learning context, and prompting strategies. Performance depended strongly on modality and model. MIDI input yielded near-ceiling scores, especially for Gemini models, whereas audio reduced accuracy across tasks, highlighting perception from waveform as the primary bottleneck. Qwen2.5-Omni generally underperformed, with the largest LogicLM deficits.

ZS vs. FS. Table 1 shows minimal differences when comparing the accuracies of ZS and FS conditions. FS tended to help Syncopation in audio (e.g., Gemini Pro from ∼25% ZS to ∼65% FS in Standalone/CoT), but this trend was not reliable across models or tasks.

Prompting strategies. Prompting effects varied by task. For Syncopation, CoT offered modest gains in audio, while LogicLM was only beneficial with MIDI (Gemini reaching 95–100%). For Transposition, Standalone and CoT prompts worked best, while LogicLM reduced accuracy. Chord ID was trivial in Standalone/CoT but collapsed with LogicLM-audio due to schema fragility. Overall, neuro-symbolic prompting helped only when inputs were symbolic and formatting was reliable.

Per-task GLMMs: Assessing modality differences. Given that reasoning style (Standalone, CoT, or LogicLM) and shot learning setting (few-shot vs. zero-shot) had minimal effects, we statistically tested the effects of model and modality on performance using

Table 1: Accuracy of multimodal LLMs on three music perception tasks: syncopation scoring, transposition detection, and chord quality identification. Results are reported for audio and MIDI inputs under three prompting strategies (Standalone, CoT, LogicLM) and zero-shot (ZS) vs few-shot (FS) learning conditions. **Bold** highlights best performance per task/shot/modality (underlined shows second best). A systematic gap between modalities is seen: MIDI inputs generally lead to higher accuracies and clearer prompting effects compared to audio. The bottom row represents chance performance.

			Syncopation			Transposition			Chord ID		
Mod.	Shot	Cond.	Flash	Pro	Qwen	Flash	Pro	Qwen	Flash	Pro	Qwen
Audio	ZS	Stand.	30.00	25.00	20.00	55.56	94.74	75.00	31.82	**47.73**	31.82
		CoT	**35.00**	25.00	20.00	76.92	**95.00**	65.00	31.82	43.18	31.82
		LogicLM	20.00	20.00	20.00	65.00	80.00	50.00	11.36	18.18	6.82
	FS	Stand.	31.58	63.16	40.00	**94.74**	90.00	90.00	25.00	40.91	31.82
		CoT	40.00	**65.00**	40.00	63.16	90.00	60.00	25.00	**52.27**	34.09
		LogicLM	40.00	55.00	20.00	60.00	90.00	35.00	6.82	13.64	18.18
MIDI	ZS	Stand.	84.21	95.00	25.00	**100.00**	**100.00**	85.00	50.00	97.73	22.73
		CoT	94.74	**100.00**	35.00	95.00	**100.00**	20.00	**100.00**	**100.00**	25.00
		LogicLM	90.00	80.00	20.00	**100.00**	**100.00**	10.00	93.18	**100.00**	**100.00**
	FS	Stand.	88.89	**100.00**	35.00	**100.00**	**100.00**	90.00	70.45	**100.00**	29.55
		CoT	95.00	**100.00**	25.00	**100.00**	**100.00**	60.00	97.73	**100.00**	29.55
		LogicLM	**100.00**	95.00	25.00	**100.00**	**100.00**	15.00	**100.00**	**100.00**	**100.00**
	Chance			20.00			50.00			25.00	

GLMMs for each of the three tasks. Type III Wald χ^2 tests revealed a consistent and highly significant interaction between Model and Modality across all tasks: Chord Quality ($\chi^2(2) = 62.22$, $p < .001$), Syncopation ($\chi^2(2) = 62.11$, $p < .001$), and Transposition ($\chi^2(2) = 18.22$, $p < .001$). This confirms that the performance gap between audio and MIDI inputs is consistent across music perception tasks, but varies significantly depending on the specific model being evaluated. These interactions, visualized in Figure 2, provide a more granular understanding of each model's strengths and weaknesses.

For the Syncopation (Figure 2A) and Chord Quality (Figure 2C) tasks, the Gemini models (Flash and Pro) demonstrate a statistically significant modality gap, with the near-ceiling performance on MIDI inputs plummeting dramatically for audio stimuli (ps $< .001$). Gemini Pro is the strongest performer in the audio condition for all tasks, though its accuracy remains below 50% for Syncopation Scoring and Chord Quality ID. The interaction in the Syncopation task is particularly revealing. While the Gemini models show a large performance drop from MIDI to audio, the Qwen2.5-Omni model shows no significant difference between the two modalities ($z = -0.17$, $p = .865$). However, this is not due to strong audio performance, but rather to its equally poor performance on the symbolic MIDI data, indicating a failure in the core reasoning for that task, independent of the input modality.

The Transposition task presents the most compelling evidence of genuine audio perception. Here, the interaction effect is driven by Gemini Pro's remarkable success when ana-

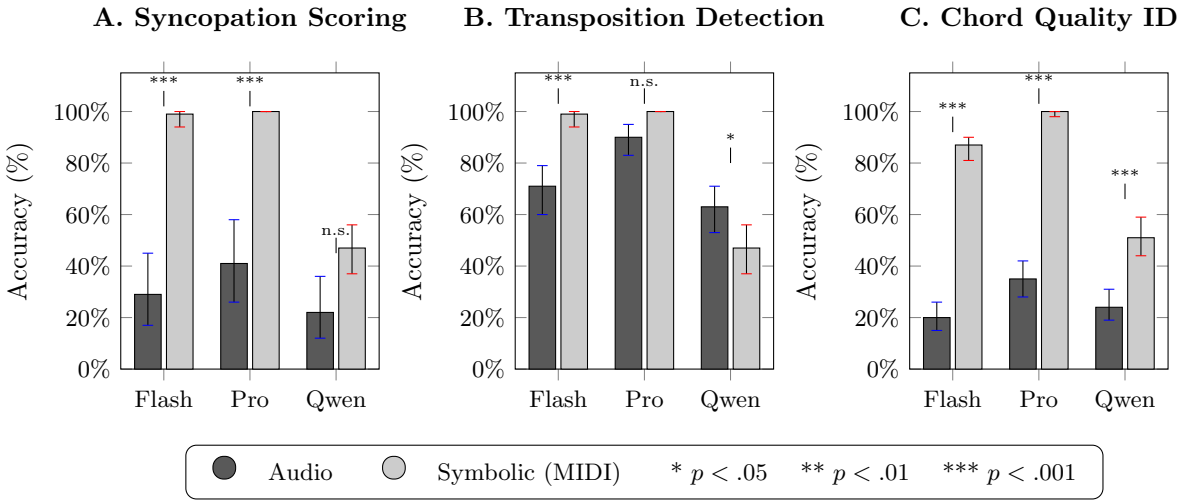

Figure 2: Model performance by modality, estimated from per-task GLMMs. Bars show estimated marginal mean accuracy with 95% confidence intervals. Significance brackets denote the results of pairwise post-hoc tests comparing Audio and MIDI performance per model. A (Syncopation Scoring): we see significant effects of modality for both Gemini Flash and Pro ($p < .001$), but not for Qwen ($p = .864$). B (Transposition Detection): we see significant effects of modality for Gemini Flash ($p < .001$) and Qwen ($p < .05$), and see that the high accuracies of Gemini Pro for both Audio and MIDI result in no significant differences between the two modalities ($p = .899$). C (Chord Quality ID): we see significant effects of modality for all three models ($p < .001$). n.s., not significant.

lyzing audio. Post-hoc comparisons revealed that while Flash and Qwen still exhibited a significant modality gap, the difference between Gemini Pro's audio and MIDI performance was not statistically significant ($z = -0.13$, $p = .899$). As visualized in the Transposition plot (Figure 2B), Gemini Pro achieves up to 95% accuracy on audio inputs, effectively closing the perceptual gap and performing on par with its MIDI accuracy. This singular achievement highlights that, for certain relational reasoning tasks like melody comparison, state-of-the-art models are beginning to bridge the divide between symbolic reasoning and true audio-native understanding (i.e., real "listening").

LogicLM input quality and response accuracy. Figure 3 diagnoses the source of the performance gaps observed in Table 1 by disentangling transcription fidelity from downstream reasoning. This visualization reveals that the primary bottleneck for audio-based tasks is a failure in perception rather than downstream reasoning. The most illuminating example of this is in Chord Quality Identification (pastel-colored points). For audio inputs, the models exhibit a complete perceptual breakdown, with F1-scores clustering below 0.25. This indicates an inability to correctly identify the constituent pitches of a chord from the raw waveform. Consequently, with unreliable symbolic input, the models' reasoning collapses, yielding final accuracy scores that hover near chance level, corroborating the poor performance seen in Table 1. In stark contrast, when provided with clean MIDI data, the models achieve near-perfect F1-scores and accuracy, demonstrating that their capacity for

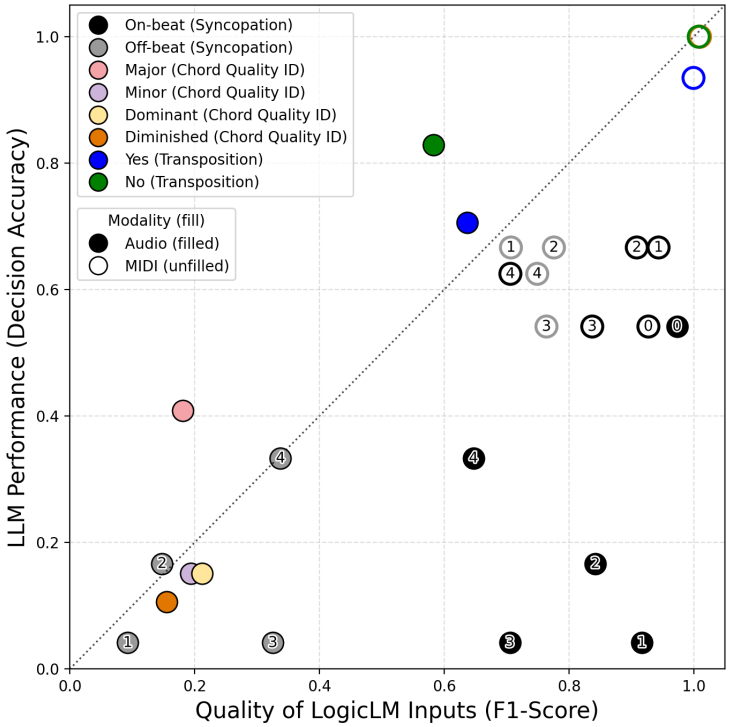

Figure 3: Relationship between the quality of LogicLM inputs (F1-scores, x-axis) and final decision accuracy (y-axis). Each point represents a specific task category, pooled across all models and shot settings. Circle fill encodes modality (Audio = filled, MIDI = unfilled) and color encodes the task-specific category (on- and off-beats for syncopation; chord qualities for chord ID; Yes/No responses for transposition). Syncopation circles are further partitioned by syncopation level. The plot highlights the perceptual bottleneck: the points pertaining to audio consistently show lower F1-scores and accuracy compared to MIDI (most overlap at (1,1)).

symbolic reasoning about harmony is intact but is hampered when fed unreliable perceptual input.

The Syncopation Scoring task offers a more nuanced illustration of this perceptual bottleneck. The plot uniquely separates the F1-scores for on-beat events (black circles) from off-beat events (grey circles), and shows individual points for each level of syncopation that we test. For MIDI inputs, both on-beat and off-beat transcription F1-scores are exceptionally high (> 0.7), leading to high decision accuracy. However, for audio inputs, a critical divergence appears: while the models are fairly successful at transcribing the rhythmically simple on-beats (F1-scores between 0.6 and 0.95), they are largely unable to detect the crucial syncopated off-beats (F1-scores below 0.4). Since the final syncopation score is entirely dependent on counting these off-beats, the low perceptual fidelity for this specific feature directly causes the low overall accuracy for the audio-based syncopation task. Missing off-beat events is crucial beyond this benchmark, as syncopation is a fundamental feature of music that shapes perception, underlies the feeling of groove, and modulates music reward (Matthews et al., 2019, 2020). Thus, the models can estimate the pulse, but cannot hear the syncopation.

Finally, the Transposition Detection task stands out as the most robust in the audio modality, a finding consistent across analyses. For audio inputs (blue and green points), the models achieve F1-scores in the 0.6–0.7 range, which, while imperfect, are substantially better than in other audio tasks. This moderately successful perception is sufficient to drive decision accuracy to relatively high levels (approximately 70–80%). This suggests that extracting melodic contour and relative pitch intervals (the core components of transposition) is a more tractable perceptual task for current models than the precise onset detection required for syncopation or the harmonic parsing needed for chord identification. Overall, the plot compellingly argues that future progress in musical AI will depend less on enhancing the abstract reasoning of LLMs and more on improving the robustness of their audio front-ends to reliably transcribe the foundational elements of music.

4. Discussion

Our findings converge on a simple but consequential claim: multimodal LLMs reason effectively over symbolic music data, yet still fail to truly "listen". LLMs, especially Gemini models, reached near-ceiling with MIDI, and LogicLM behaved as intended once schema adherence was met. Replacing MIDI with audio sharply reduced accuracy, especially for Syncopation Scoring and Chord Quality ID under LogicLM, implicating transcription/onset tracking and pitch-salience as the primary bottlenecks (Weck et al., 2024; Ghosh et al., 2025a; Marták et al., 2025). Reasoning strategies (CoT, LogicLM) did not compensate for upstream hearing errors (Zhifei et al., 2025). This modality gap matters because people experience music through audio, not symbolic proxies. Symbolic formats strip away the features making music meaningful (micro-timing, articulation, expressive nuance) so LLM ceiling performance on MIDI should not be mistaken for audio-native competence (Ayyildiz et al., 2025; Groves et al., 2025). Our GLMM analysis provides statistical support for these observations, but more importantly, reveals a significant Model $\times$ Modality interaction throughout. That is, although models performed better on MIDI than on audio, the magnitude of this difference varied across models and music perception tasks. This finding complicates a simple narrative of universal audio failure. It demonstrates that the severity of the perceptual bottleneck is model and task-dependent. For instance, the Qwen2.5-Omni model's failure on the Syncopation task was unique in that its performance was equally poor on MIDI and Audio data, suggesting a more fundamental deficit in its symbolic reasoning capabilities for that task, rather than just a perceptual one. In contrast, Gemini Pro's success in the Melody Transposition task, where it statistically closed the performance gap between audio and MIDI, stands as a crucial proof-of-concept. It suggests that for tasks reliant on global melodic features, state-of-the-art architectures are on the cusp of achieving true audio-native competence, even if they fall short elsewhere.

In sum, current multimodal LLMs reason symbolically but lack fully accurate audio-native competence: the ability to process songs from audio files to answer structured questions. We suggest that progress will depend on stronger audio front-ends and propagation of uncertainty into downstream solvers. In the current state-of-the-art, symbolic reasoning layers collapse due to small perceptual errors. LLMs that acquire genuine understanding could also be music education (Jin et al., 2025) and user-centric music analysis tools (Urrego-Gómez et al., 2025; Carone and Ripollés, 2024), enabling interactive systems that can teach musical structure and foster deeper engagement with personal music listening.

References

Ceren Ayyildiz, Andrew J. Milne, Muireann Irish, and Steffen A. Herff. Micro-variations in timing and loudness affect music-evoked mental imagery. *Scientific Reports*, 15 (1):30967, 2025. doi: 10.1038/s41598-025-12604-4. URL https://doi.org/10.1038/s41598-025-12604-4. Published: 2025-08-22.

Brandon J. Carone and Pablo Ripollés. Soundsignature: What type of music do you like? In *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*, pages 1–10, 2024.

Brandon J. Carone, Iran R. Roman, and Pablo Ripollés. The MUSE benchmark: Probing music perception and auditory relational reasoning in audio llms. *arXiv preprint arXiv:2510.19055*, 2025a.

Brandon James Carone, Iran R. Roman, and Pablo Ripollés. Evaluating multimodal large language models on core music perception tasks. In *39th Conference on Neural Information Processing Systems Workshop: AI for Music*, 2025b.

Yunfei Chu, Jin Xu, Xiaohuan Zhou, Qian Yang, Shiliang Zhang, Zhijie Yan, Chang Zhou, and Jingren Zhou. Qwen-audio: Advancing universal audio understanding via unified large-scale audio-language models. *arXiv preprint arXiv:2311.07919*, 2023.

Yunfei Chu, Jin Xu, Qian Yang, Haojie Wei, Xipin Wei, Zhifang Guo, Yichong Leng, Yuanjun Lv, Jinzheng He, and Junyang Lin. Qwen2-audio technical report. *arXiv preprint arXiv:2407.10759*, 2024.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, and Evan Rosen. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Diana Deutsch. Music recognition. *Psychol Rev*, 76(3):300–7, 1969.

Jay W. Dowling. Scale and contour: Two components of a theory of memory for melodies. *Psychological Review*, 85(4):341–354, 1978.

Jay W. Dowling and Diane S. Fujitani. Contour, interval, and pitch recognition in memory for melodies. *Journal of the Acoustical Society of America*, 49(2, Pt. 2):524–531, 1971.

Sreyan Ghosh, Arushi Goel, Lasha Koroshinadze, Sang gil Lee, Zhifeng Kong, Joao Felipe Santos, Ramani Duraiswami, Dinesh Manocha, Wei Ping, Mohammad Shoeybi, and Bryan Catanzaro. Music flamingo: Scaling music understanding in audio language models, 2025a. URL https://arxiv.org/abs/2511.10289.

Sreyan Ghosh, Zhifeng Kong, Sonal Kumar, S Sakshi, Jaehyeon Kim, Wei Ping, Rafael Valle, Dinesh Manocha, and Bryan Catanzaro. Audio flamingo 2: An audio-language model with long-audio understanding and expert reasoning abilities. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 19358–19405. PMLR, 13–19 Jul 2025b.

Karleigh Groves, Morwaread Mary Farbood, Brandon Carone, Pablo Ripollés, and Arianna Zuanazzi. Acoustic features of instrumental movie soundtracks elicit distinct and mostly non-overlapping extra-musical meanings in the mind of the listener. *Scientific reports*, 15 (1):2327, 2025.

Andrea R Halpern and James C Bartlett. *Memory for melodies*, pages 233–258. Springer, 2010.

Lingxi Jin, Baicheng Lin, Mengze Hong, Kun Zhang, and Hyo-Jeong So. Exploring the impact of an llm-powered teachable agent on learning gains and cognitive load in music education, 2025. URL https://arxiv.org/abs/2504.00636.

Zhifeng Kong, Arushi Goel, Rohan Badlani, Wei Ping, Rafael Valle, and Bryan Catanzaro. Audio flamingo: A novel audio language model with few-shot learning and dialogue abilities. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 25125–25148. PMLR, 21–27 Jul 2024.

Sonal Kumar, Šimon Sedláček, Vaibhavi Lokegaonkar, Fernando López, Wenyi Yu, Nishit Anand, Hyeonggon Ryu, Lichang Chen, Maxim Plička, and Miroslav Hlaváček. Mmau-pro: A challenging and comprehensive benchmark for holistic evaluation of audio general intelligence. *arXiv preprint arXiv:2508.13992*, 2025.

Edward W. Large, Jorge A. Herrera, and Marc J. Velasco. Neural networks for beat perception in musical rhythm. *Frontiers in Systems Neuroscience*, Volume 9 - 2015, 2015.

Edward W Large, Iran Roman, Ji Chul Kim, Jonathan Cannon, Jesse K Pazdera, Laurel J Trainor, John Rinzel, and Amitabha Bose. Dynamic models for musical rhythm perception and coordination. *Frontiers in Computational Neuroscience*, 17:1151895, 2023.

Shansong Liu, Atin Sakkeer Hussain, Chenshuo Sun, and Ying Shan. Music understanding llama: Advancing text-to-music generation with question answering and captioning. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 286–290. IEEE, 2024.

Yinghao Ma, Siyou Li, Juntao Yu, Emmanouil Benetos, and Akira Maezawa. Cmi-bench: A comprehensive benchmark for evaluating music instruction following. *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR 2025)*, 2025a.

Ziyang Ma, Yinghao Ma, Yanqiao Zhu, Chen Yang, Yi-Wen Chao, Ruiyang Xu, Wenxi Chen, Yuanzhe Chen, Zhuo Chen, and Jian Cong. Mmar: A challenging benchmark for deep reasoning in speech, audio, music, and their mix. *arXiv preprint arXiv:2505.13032*, 2025b.

Elizabeth Hellmuth Margulis. *On Repeat: How Music Plays the Mind*. Oxford University Press, 2013.

Lukáš Samuel Marták, Patricia Hu, and Gerhard Widmer. Sound and music biases in deep music transcription models: a systematic analysis. *EURASIP Journal on Audio, Speech,*

and Music Processing, 2025. ISSN 1687-4722. doi: 10.1186/s13636-025-00428-z. URL https://doi.org/10.1186/s13636-025-00428-z. Published online: 2025-12-11.

Tomas E Matthews, Maria AG Witek, Ole A Heggli, Virginia B Penhune, and Peter Vuust. The sensation of groove is affected by the interaction of rhythmic and harmonic complexity. *PLoS One*, 14(1):e0204539, 2019.

Tomas E Matthews, Maria AG Witek, Torben Lund, Peter Vuust, and Virginia B Penhune. The sensation of groove engages motor and reward networks. *NeuroImage*, 214:116768, 2020.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. Logic-LM: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824, Singapore, 2023. Association for Computational Linguistics.

S Sakshi, Utkarsh Tyagi, Sonal Kumar, Ashish Seth, Ramaneswaran Selvakumar, Oriol Nieto, Ramani Duraiswami, Sreyan Ghosh, and Dinesh Manocha. MMAU: A massive multi-task audio understanding and reasoning benchmark. In *The Thirteenth International Conference on Learning Representations*, 2025.

Changli Tang, Wenyi Yu, Guangzhi Sun, Xianzhao Chen, Tian Tan, Wei Li, Lu Lu, Zejun MA, and Chao Zhang. Salmonn: Towards generic hearing abilities for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Isabel Urrego-Gómez, Simon Colton, and Iran R Roman. Vibe sorcery: Integrating emotion recognition with generative music for playlist curation. In *International Society for Music Information Retrieval: 1st Workshop on Large Language Models for Music & Audio*, 2025.

Benno Weck, Ilaria Manco, Emmanouil Benetos, Elio Quinton, György Fazekas, and Dmitry Bogdanov. Muchomusic: Evaluating music understanding in multimodal audio-language models. In *Proceedings of the 25th International Society for Music Information Retrieval Conference (ISMIR)*, 2024.

Junda Wu, Zachary Novack, Amit Namburi, Hao-Wen Dong, Carol Chen, Jiaheng Dai, and Julian McAuley. Futga-mir: Enhancing fine-grained and temporally-aware music understanding with music information retrieval. In *ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 2025.

Jin Xu, Zhifang Guo, Jinzheng He, Hangrui Hu, Ting He, Shuai Bai, Keqin Chen, Jialin Wang, Yang Fan, and Kai Dang. Qwen2. 5-omni technical report. *arXiv preprint arXiv:2503.20215*, 2025.

Qian Yang, Jin Xu, Wenrui Liu, Yunfei Chu, Ziyue Jiang, Xiaohuan Zhou, Yichong Leng, Yuanjun Lv, Zhou Zhao, Chang Zhou, and Jingren Zhou. AIR-bench: Benchmarking large audio-language models via generative comprehension. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1979–1998, Bangkok, Thailand, 2024. Association for Computational Linguistics.

Yongyi Zang, Sean O'Brien, Taylor Berg-Kirkpatrick, Julian McAuley, and Zachary Novack. Are you really listening? boosting perceptual awareness in music-qa benchmarks. *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR 2025)*, 2025.

Xie Zhifei, Mingbao Lin, Zihang Liu, Pengcheng Wu, Shuicheng Yan, and Chunyan Miao. Audio-reasoner: Improving reasoning capability in large audio language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 23840–23862, 2025.

Supplementary Materials

S1. Stimuli

Stimuli are original musical recordings created by a real human musician in Logic Pro X using a 2021 16" MacBook Pro (Apple M1 Pro chip), an Apollo Twin X audio interface, and Yamaha HS8 monitors. Stimuli were recorded on electric guitar (PRS McCarty Hollowbody II, Schecter Solo-6), piano (Arturia KeyLab Essential Mk3 MIDI controller with Analog Lab V software instruments), and drums (Roland TD-17 electronic kit with Superior Drummer 3 plugin). Guitar recordings were processed with Neural DSP plugins (Tim Henson Archetype, Cory Wong Archetype).

Additional excerpts were reserved for few-shot prompting (2 for syncopation, 2 for transposition, and 4 for chord ID, one per chord class) and excluded from testing.

You can access the stimuli used in this experiment on The MUSE Benchmark Github page.

The stimuli used for the Transposition Detection task can be found here and all of them have the melody number, key, and tempo in the filename (e.g., M1_EbMaj_90.wav).

The stimuli used for the Syncopation Scoring task can be found here and all of them have Sync in the name, along with the syncopation level number (e.g., NoSync_A, Sync2_B).

The stimuli used for the Chord Quality Identification task can be found here. The chords are named by number, and you can find the mapping in Table S1 below.

Table S1: Mapping of chord roots and qualities to numerical identifiers (1.wav–48.wav).

Root	Diminished	Dominant	Major	Minor
Ab	1	2	3	4
A	5	6	7	8
Bb	9	10	11	12
B	13	14	15	16
C	17	18	19	20
Db	21	22	23	24
D	25	26	27	28
Eb	29	30	31	32
E	33	34	35	36
F	37	38	39	40
Gb	41	42	43	44
G	45	46	47	48

S2. System Instructions

1a) Syncopation — Standalone

> "You are an expert music transcription AI participating in a multi-turn reasoning experiment.

You will be given one short audio excerpt of a drum set per trial. Your task is to focus only on the kick and snare drums. The hi-hat plays constant 8th notes, acting as a metronome. Count the total number of kicks and snare hits that fall on off-beats.

Valid multiple-choice responses are:

A. 0 (No Syncopation)
B. 2 (Low Syncopation)
C. 4 (Medium-Low Syncopation)
D. 6 (Medium-High Syncopation)
E. 8 (High Syncopation)

End with exactly one line:
Final Answer: X
"

1b) Syncopation — Chain-of-Thought (CoT)

"You are an expert music transcription AI participating in a multi-turn reasoning experiment.

You will be given one short audio excerpt of a drum set per trial. Your task is to focus only on the kick and snare drums. The hi-hat plays constant 8th notes, acting as a metronome. Count the total number of kicks and snare hits that fall on off-beats. On-beats are the main pulses (beats 1, 2, 3, and 4) and off-beats are the "ands" in between. Ignore the on-beats and ignore the hi-hat.

Valid multiple-choice responses are:

A. 0 (No Syncopation)
B. 2 (Low Syncopation)
C. 4 (Medium-Low Syncopation)
D. 6 (Medium-High Syncopation)
E. 8 (High Syncopation)

After any reasoning, end with exactly one line:
Final Answer: X
"

1c) Syncopation — LogicLM

"You are an expert music transcription AI participating in a multi-turn reasoning experiment.

Your task is to transcribe the onsets of ONLY the kick and snare drums into the format:
`rhythm(identifier, [list_of_onsets]).`

- The 'identifier' is the filename of the audio.
- The 'list_of_onsets' is a comma-separated list of integers from 1 to 32.

- The rhythm is on a 4-bar grid, quantized to 8th notes (numbered 1 to 32). All odd numbers are on-beats, and all even numbers are off-beats.
- The hi-hat plays constant 8th notes, acting as a metronome. On-beats are the main pulses (beats 1, 2, 3, and 4 of each bar) and off-beats are the 'ands' in between.

Grid: The excerpt is 4 bars quantized to 8th notes → 32 slots numbered 1–32.

Within each bar (8 slots): 1,3,5,7 = on-beats (beats 1–4). 2,4,6,8 = off-beats ("&"s).

Across bars: slot = 8×(bar-1) + local_slot.

Beat positions across 4 bars:
• Beat 1 → 1, 9, 17, 25
• Beat 2 → 3, 11, 19, 27
• Beat 3 → 5, 13, 21, 29
• Beat 4 → 7, 15, 23, 31

Off-beats ("&"s):
• &1 → 2, 10, 18, 26
• &2 → 4, 12, 20, 28
• &3 → 6, 14, 22, 30
• &4 → 8, 16, 24, 32

Output format:
`rhythm(identifier.wav, [n1, n2, ..., nK])` where each `n` is an integer in 1–32.

Example of format where the kicks are on beats 1 and 3 in each bar, and the snare hits are on beats 2 and 4 in each bar (all played on the on-beats):
`rhythm(example.wav, [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31])`

Output your answer of symbolic code as a single line of plain text without code fences or explanations. After your transcription, an external tool will score it, and you will answer a question based on that score."

2a) Transposition Detection — Standalone

"You are an expert melody transcription AI participating in a multi-turn reasoning experiment.

You will be given two short monophonic audio melodies per trial.
Your job is to decide whether they represent the SAME melody up to TRANSPOSITION (i.e., identical shape/intervals but possibly in different keys).

Valid responses are exactly one of:
"Yes, these are the same melody."
"No, these are not the same melody."

Respond with exactly one of the two phrases and nothing else.”

2b) Transposition Detection — Chain-of-Thought (CoT)

“You are an expert melody transcription AI participating in a multi-turn reasoning experiment.

You will be given two short monophonic audio melodies per trial. Your job is to decide whether they represent the SAME melody up to TRANSPOSITION (i.e., identical shape/intervals but possibly in different keys).

Definitions and constraints:
- Transposition equivalence: the two melodies have the same number of notes and the same sequence of pitch INTERVALS between successive notes (including 0 for repeated notes).
- Ignore absolute key/register, starting pitch, and tempo. Small timing variations are acceptable. If the rhythmic patterns are drastically different (e.g., note insertions/deletions or re-ordered phrases), they are most likely NOT the same melody.
- Treat repeated notes as separate events and include 0 in the interval sequence when a note repeats.
- If there are leading/trailing silences, ignore them.

Valid responses (exactly one of these strings):
“Yes, these are the same melody.”
“No, these are not the same melody.”

After any reasoning, end with exactly one line:
Final Answer: Yes, these are the same melody.
OR
Final Answer: No, these are not the same melody.”

2c) Transposition Detection — LogicLM

“You are an expert melody transcription AI participating in a multi-turn reasoning experiment.

You will be given two short monophonic audio melodies per trial. Your first task is to transcribe EACH melody into the symbolic format below, using MIDI integers for pitches. If the rhythmic sequences seem drastically different, they are most likely not the same melody.

Output format (schema):
`melody(identifier, [p1, p2, ..., pK])`

- Use the exact identifiers I provide for each trial (one per audio).
- p1..pK are integers representing MIDI pitches (e.g., C4 = 60).
- Transcribe the pitch sequence only.
- Output exactly two lines of plain text: one ‘melody(...)’ per line, in the same

order as the audios (Audio 1 line first, then Audio 2 line).
- Do not include code fences or any extra commentary.

Example (schema only; not tied to any audio):
`melody(Audio1, [60, 62, 64])`
`melody(Audio2, [65, 67, 69])`

After your transcription, a deterministic tool will analyze the two lines to decide if the melodies are transpositions (same contour, different key). You will then answer a Yes/No question based on that decision."

3a) Chord Quality Matching — Standalone

"You are an expert chord-transcription AI participating in a multi-turn reasoning experiment.

You will be given one short audio clip per trial. Each clip first plays a chord (block), then the individual notes (arpeggiation).
All chords are in ROOT POSITION.

Your task is to identify the chord QUALITY.

Valid options:
A. Major
B. Minor
C. Dominant
D. Diminished

Final Answer: X
"

3b) Chord Quality Matching — Chain-of-Thought (CoT)

"You are an expert chord-transcription assistant in a multi-turn reasoning experiment.

You will be given one short audio clip per trial containing a single chord (first block, then arpeggiated notes). All chords are in ROOT POSITION; the lowest pitch is the ROOT (treat as 0 semitones). Your task: identify the chord QUALITY by inferring pitch-class intervals above the root and ignoring octave doublings.

Valid options:
A. Major $\rightarrow$ {0,4,7}
B. Minor $\rightarrow$ {0,3,7}
C. Dominant $\rightarrow$ {0,4,7,10}
D. Diminished $\rightarrow$ {0,3,6}

Think through the identification. Once you've finished reasoning, the final line of your output should be exactly:

Final Answer: X
”

3c) Chord Quality Matching — LogicLM

“You are an expert chord-transcription assistant in a multi-turn reasoning experiment.

You will be given one short audio clip per trial containing a single chord. First the chord sounds as a block, then the notes are arpeggiated.

Your task is to transcribe the chord tones into a strict symbolic format. Use MIDI integers (0–127). Include octave doublings if you hear them. Do not add commentary.

Output format (schema):
`chord(identifier, [p1, p2, ..., pK])`

Rules:
- Use the exact identifier I provide for the trial.
- Record only the pitches you hear as MIDI integers.
- It is acceptable if the list is not sorted; a deterministic solver will normalize.
- Output EXACTLY ONE LINE of plain text with NO code fences or extra text.

Example (schema only; not tied to any audio):
`chord(Audio_X, [56, 60, 64, 67, 72, 76])`

After your line is produced, a deterministic tool will classify the chord quality (Major / Minor / Dominant / Diminished) from your symbolic line. You will then answer a multiple-choice question with: Final Answer: X”

S3. `Task Schemas and Deterministic Solvers`

Each task defines a single-line schema the model must emit verbatim. A hand-written, deterministic solver (`solver.py`) parses that line, makes the decision, and returns the minimal information needed for a constrained final answer.

Syncopation Scoring

Input: 4-bar drum loop with constant 8th-note hi-hat; we only score kick+snare.
Grid: 32 slots (8 per bar). Odd slots are on-beats; even slots are off-beats.

Schema (one line):

`rhythm(<id>, [n1, n2, ..., nK])`

Where each `n` is an integer in [1..32] (kick or snare onset).
Solver: counts off-beat onsets and maps to five categories: 0,2,4,6,8 off-beats → A–E respectively. Final answer is a single MC letter A–E.

Transposition Detection

Input: two short monophonic excerpts (guitar or piano) that are either the same melody in different keys or different melodies.

Schema (two lines, order-preserving):

```
melody(<id1>, [p1, p2, ..., pK])
melody(<id2>, [p1, p2, ..., pK])
```

Where `p*` are MIDI integers (0–127).

Solver: checks equal length and equality of adjacent-interval sequences (transposition invariance). Returns ARE / ARE NOT (transpositions). Final answer is forced to one of:

"Yes, these are the same melody."
"No, these are not the same melody."

Chord Quality Identifier

Input: a single triad or seventh chord (piano), presented as a block then arpeggiated.

Schema (one line):

```
chord(<id>, [p1, p2, ..., pK])
```

MIDI integers (0–127); octave doublings allowed.

Solver: normalizes to pitch classes, factors out the putative root, and matches the interval set to:

- Major (0, 4, 7) → A
- Minor (0, 3, 7) → B
- Dominant 7 (0, 4, 7, 10) → C
- Diminished (0, 3, 6) → D

Final answer is a single MC letter A–D.

Self-refinement (SR)

For LogicLM, we validate the line(s) with strict regex/AST checks and label errors as parse, structural, or domain. If invalid, we run up to 2 SR rounds in a separate deterministic chat (temperature=0, top_p=1, top_k=1, 256 tokens) with a fix-only prompt that:

- Echoes the prior output,
- States the specific error type/message,
- Re-states the required line(s) and constraints,
- Forbids commentary and code fences.

If the solver returns undecidable/None (e.g., empty list), we allow one extra SR pass with a synthesized parse error. This SR design follows the LogicLM self-refinement idea of using solver feedback to repair the symbolic form.

S3.1. solver.py

```
# solver.py
import re
from typing import List, Optional, Tuple, Dict

class SyncopationSolver:
    """
    A deterministic logic solver that calculates a syncopation score
    based on a simplified on-beat/off-beat rule for a 4-bar (1-32) 8th-note grid.
    """
    def __init__(self):
        self.on_beats = set()
        self.off_beats = set()

        for bar_offset in [0, 8, 16, 24]:
            self.on_beats.update([
                1 + bar_offset, 3 + bar_offset, 5 + bar_offset, 7 + bar_offset
            ])
            self.off_beats.update([
                2 + bar_offset, 4 + bar_offset, 6 + bar_offset, 8 + bar_offset
            ])

    def parse_llm_output(self, llm_text: str) -> Optional[List[int]]:
        """
        Parses the LLM's symbolic output to extract a list of onsets.
        Returns the list of integers if successful, or None if parsing fails.
        """
        match = re.search(r'rhythm\s*\(\s*[^,]+\s*,\s*\[([\d,\s]*)\]\s*\)',
            llm_text)
        if not match:
            return None
        numbers_str = match.group(1)
        if not numbers_str.strip(): # Check if the string is empty or just
            whitespace
            return []
        try:
            # Handle potential trailing commas by filtering out empty strings
                after split
            return [int(num.strip()) for num in numbers_str.split(',') if
                num.strip()]
        except ValueError:
            return None

    def score_onset(self, onset: int) -> int:
        if onset in self.off_beats:
            return 1
        return 0

    def calculate_total_score(self, onset_list: list[int]) -> int:
        if not onset_list:
```

```
            return 0
        total_score = sum(self.score_onset(onset) for onset in onset_list)
        return total_score

class TranspositionSolver:
    """
    A deterministic solver for melody transposition detection.
    Two melodies are considered transpositions if:
      - They have the same number of notes, and
      - Their interval sequences (adjacent pitch differences in semitones) are
          identical.
    Rhythm is ignored. Pitches must be integers (MIDI numbers).
    """

    MELODY_PATTERN = re.compile(
        r"melody\s*\(\s*([A-Za-z0-9_.\-]+)\s*,\s*\[\s*([^\]]*?)\s*\]\s*\)",
        flags=re.IGNORECASE
    )

    def _extract_pitches(self, pitches_str: str) -> Optional[List[int]]:
        """
        Extracts integer pitches from an arbitrary list content that may include
        parentheses or spaces, e.g. '[(60), (62), (64)]' or '60, 62,64'.
        """
        nums = re.findall(r"-?\d+", pitches_str)
        if not nums:
            return []
        try:
            return [int(n) for n in nums]
        except ValueError:
            return None

    def parse_llm_output(self, llm_text: str) -> Optional[List[Dict[str,
        List[int]]]]:
        """
        Parses any 'melody(ID, [ ... ])' lines found in the LLM's output, in order.
        Returns a list of dicts: [{'id': <ID>, 'pitches': [..]}, ...]
        or None if nothing parseable is found.
        """
        if not llm_text:
            return None

        text = llm_text.replace("```", "").replace("`", "").strip()

        melodies = []
        for m in self.MELODY_PATTERN.finditer(text):
            ident = m.group(1)
            plist_str = m.group(2)
            pitches = self._extract_pitches(plist_str)
            if pitches is None:
                return None
```

```
            melodies.append({"id": ident, "pitches": pitches})

        return melodies or None

    def _intervals(self, pitches: List[int]) -> List[int]:
        return [pitches[i+1] - pitches[i] for i in range(len(pitches) - 1)]

    def are_transpositions(self, p1: List[int], p2: List[int]) -> Optional[bool]:
        """
        Returns True/False if a decision is possible, or None if inputs are
            degenerate.
        Policy:
          - Require same length (>0). If lengths differ, return False.
          - If length == 1 on both, return True (single note can be transposed
              anywhere).
          - Otherwise compare interval sequences.
        """
        if p1 is None or p2 is None:
            return None
        if len(p1) == 0 and len(p2) == 0:
            return None
        if len(p1) != len(p2):
            return False
        if len(p1) == 1: # single-note melodies
            return True

        return self._intervals(p1) == self._intervals(p2)

    def decide_same_melody(self, llm_text: str) -> Optional[bool]:
        """
        Convenience: parse two melodies from LLM output and decide True/False.
        Returns None if fewer than 2 melodies parsed or if undecidable.
        """
        parsed = self.parse_llm_output(llm_text)
        if not parsed or len(parsed) < 2:
            return None
        p1 = parsed[0]["pitches"]
        p2 = parsed[1]["pitches"]
        return self.are_transpositions(p1, p2)


# ---------- Chord Quality (deterministic) ----------

class ChordQualitySolver:
    """
    Deterministic chord-quality classifier for LogicLM.
    Expects ONE schema line produced by the LLM:
        chord(identifier, [p1, p2, ..., pK])

    Behavior:
    - Parses the line and extracts MIDI integers (duplicates allowed).
```

```
- Sorts pitches, treats the lowest as the root, and computes (p - root) % 12.
- Deduplicates + sorts the pitch-class intervals and matches one of the
  four target fingerprints:
    (0,4,7)     -> ("Major", "A")
    (0,3,7)     -> ("Minor", "B")
    (0,4,7,10)  -> ("Dominant", "C")
    (0,3,6)     -> ("Diminished", "D")

Returns:
    (identifier, quality_str, letter) or None if undecidable.
"""
CHORD_PATTERN = re.compile(
    r"chord\s*\(\s*([A-Za-z0-9_.\-]+)\s*,\s*\[\s*([^\]]*?)\s*\]\s*\)",
    flags=re.IGNORECASE
)

QUALITY_BY_PCS: Dict[Tuple[int, ...], Tuple[str, str]] = {
    (0, 4, 7):     ("Major", "A"),
    (0, 3, 7):     ("Minor", "B"),
    (0, 4, 7, 10): ("Dominant", "C"),
    (0, 3, 6):     ("Diminished", "D"),
}

def _extract_pitches(self, pitches_str: str) -> Optional[List[int]]:
    """
    Robust integer pull; accepts '60,64,67', '[(60), 64, 67]', etc.
    Returns list[int] or None if malformed.
    """
    nums = re.findall(r"-?\d+", pitches_str or "")
    try:
        return [int(n) for n in nums]
    except Exception:
        return None

def parse_llm_output(self, llm_text: str) -> Optional[Dict[str, List[int]]]:
    """
    Parse the first chord(...) line found. Returns {'id': <ID>, 'pitches':
        [...]}
    or None if not found / ill-formed.
    """
    if not llm_text:
        return None
    text = llm_text.replace("```", "").strip()
    m = self.CHORD_PATTERN.search(text)
    if not m:
        return None
    ident = m.group(1)
    pitches = self._extract_pitches(m.group(2))
    if pitches is None:
        return None
    return {"id": ident, "pitches": pitches}
```

```python
def _normalize_to_pcs(self, pitches: List[int]) -> Optional[Tuple[int, ...]]:
    """
    Sort, take lowest as root, compute pitch-class intervals modulo 12,
    then deduplicate and sort.
    """
    if not pitches:
        return None
    root = min(pitches)
    pcs = tuple(sorted({(p - root) % 12 for p in pitches}))
    return pcs

def classify_quality(self, pitches: List[int]) -> Optional[Tuple[str, str]]:
    """
    Map normalized pitch-class interval set to (quality, letter).
    """
    pcs = self._normalize_to_pcs(pitches)
    if pcs is None:
        return None
    return self.QUALITY_BY_PCS.get(pcs)

def decide_quality(self, llm_text: str) -> Optional[Tuple[str, str, str]]:
    """
    End-to-end convenience used by the runner:
      - parse -> classify
    Returns (identifier, quality_str, letter) or None if undecidable.
    """
    parsed = self.parse_llm_output(llm_text)
    if not parsed:
        return None
    ident = parsed["id"]
    result = self.classify_quality(parsed["pitches"])
    if result is None:
        return None
    quality, letter = result
    return ident, quality, letter
```