Evaluating Multimodal Large Language Models on Core Music Perception Tasks

Brandon J. Carone **Pablo Ripollés**
Department of Psychology, Music and Audio Research Laboratory
New York University
bcarone@nyu.edu | pripolles@nyu.edu

Iran R. Roman
Department of Electronic Engineering and Computer Science
Queen Mary University of London
i.roman@qmul.ac.uk

Abstract

Multimodal Large Language Models (LLMs) claim "musical understanding" via evaluations that 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, Transposition Detection, and Chord Quality Identification. Moreover, we separate three sources of variability: (i) perceptual limitations (audio vs. MIDI inputs), (ii) exposure to examples (zero- vs. few-shot manipulations), and (iii) reasoning strategies (Standalone, CoT, LogicLM). For the latter we adapt LogicLM, a framework combining LLMs with symbolic solvers to perform structured reasoning, to music. Results reveal a clear perceptual gap: models perform near ceiling on MIDI but show 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. Overall, current systems reason well over symbols (MIDI) but do not yet "listen" reliably from audio. Our method and dataset make the perception–reasoning boundary explicit and offer actionable guidance for building robust, audio-first music systems.

1 Introduction

Multimodal foundation models like Qwen2.5-Omni [1] and Gemini 2.5 [2] now claim "musical understanding," yet their audio capabilities remain poorly characterized. While benchmarks like AIR-Bench [3], MMAR [4], MMAU [5], and MMAU-Pro [6], CMI-Bench [7], RUListening [8], and FUTGA-MIR [9] assess music through classification and captioning tasks, they cannot distinguish whether models genuinely perceive musical structure or rely on superficial spectral patterns. Audio-language models like SALMONN [10], Qwen-Audio [11], and Audio Flamingo 2 [12] achieve strong performance on speech and sound recognition but remain untested on the relational properties naturally embedded in music. These abilities are critical to deliver the next generation of technologies for tasks such as playlist recommendation/generation[13–16] and musical preference modeling [17].

We address this gap by testing three fundamental musical abilities that require structural understanding rather than surface recognition. Syncopation scoring demands sensitivity to rhythmic expectation violations and metric displacement [18, 19]. Transposition recognition requires melody identification invariant to absolute pitch [20–23], the core perceptual skill underlying human melodic recognition across keys and timbres [24, 25]. Chord quality identification necessitates interval pattern recognition rather than absolute frequency matching. These tasks probe the structural understanding

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: AI for Music.

that characterizes human music cognition and perception but remains absent from existing audio benchmarks.

To isolate perception from reasoning, we adapt LogicLM [26], where models serve as Perceptual Formulators generating machine-checkable symbolic schemas that deterministic solvers execute, to enhance logical reasoning and problem-solving accuracy. This approach prevents "unfaithful reasoning" [26] where correct answers mask flawed perceptual analysis. We compare audio vs. MIDI processing to measure the perception bottleneck absent in existing evaluations. Our benchmark reveals that current multimodal LLMs reason effectively over musical symbols but fail to reliably parse audio, a fundamental limitation for real-world music applications.

2 Methods

2.1 Tasks

Syncopation Scoring 20 rhythmic excerpts (8 secs each) at 120 BPM, performed on hi-hat, kick and snare drums. The hi-hat maintained constant eighth notes, while kick and snare varied across on-beats (quarter notes) and off-beats (intervening eighths). The task was to compute a Syncopation Score by counting off-beat kick/snare events and mapping the total to a categorical score (0, 2, 4, 6, or 8), following Large et al. [18]. Stimuli systematically covered the full syncopation range.

Transposition Detection Models were presented with 20 excerpt pairs (mean duration $\approx$ 9 s). In each pair, the second excerpt was either the same melody transposed to another key or a different melody. The task was to decide whether the two excerpts represented the same melody. Half the trials were matches, half mismatches. Stimuli (guitar or piano) varied in tempo, key, meter, and length.

Chord Quality Identification Models were given 44 excerpts (9 s each, 120 BPM), each consisting of a single chord presented first as a block and then as an ascending arpeggiation. All chords were in root position and played on piano. The task was to classify each chord as one of four options: A) Major (root + major 3rd + perfect 5th), B) Minor (root + minor 3rd + perfect 5th), C) Dominant (root + major 3rd + perfect 5th + minor 7th), or D) Diminished (root + minor 3rd + diminished 5th).

2.2 Stimuli

Stimuli are original musical recordings created by a real human musician, and are originally from The MUSE Benchmark [27]. Please see Appendix A for more information on stimuli.

2.3 Implementation

We adapt the LogicLM pipeline [26] to music, comparing three prompting strategies: Standalone, Chain-of-Thought (CoT), and LogicLM (symbolic reasoning with self-refinement). Trials are independent: each begins in a fresh chat session with no history carryover across trials or tasks. All strategies use identical task-specific system instructions specifying rules and output schema. We factorially cross three factors: modality (audio vs. symbolic/MIDI), reasoning (Standalone vs. CoT vs. LogicLM), and shot setting (ZS=zero-shot vs. FS=few-shot), yielding 12 conditions per task.

Standalone, CoT, LogicLM. Standalone elicits only the final categorical response (e.g., "Yes"/"No"; "C. Dominant"). CoT elicits brief intermediate reasoning, followed by a final answer. LogicLM requires symbolic transcription (e.g., rhythmic onset grid or pitch-interval list), parsed by a deterministic solver (see `solver.py`; see Appendix C). On schema violations (e.g., malformed syntax), a self-refinement loop requests the model to repair its output, mirroring Pan et al. [26]. System instructions for each task and condition can be found in Appendix B.

Zero-shot vs. few-shot. ZS presents only the instructions and stimuli. FS adds worked examples in the trial history (2 for syncopation; 2 for transposition; 4 for chord ID, one per class), each paired with the correct solution. Examples appear only in-context for that trial and are excluded from evaluation.

Per-task modularity. Each task has an isolated script, stimuli, and outputs; prompt information does not leak across strategies. All conditions follow the same evaluation structure for direct comparison.

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) 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		

Audio vs. MIDI modality. For the MIDI modality, audio items are re-performed on a MIDI keyboard and exported to `.txt` via a custom script using `mido`. Prompts swap "you will hear…" for "you will be given MIDI data…" while keeping the same required schema as in audio runs. All LLM outputs are regex-parsed to extract the final line (e.g., "Final Answer: B" or "Yes"). For LogicLM, the symbolic output is scored by the solver's decision. All trials are randomized and logged with model configuration, trial IDs, raw outputs, parsed responses, and evaluation results.

2.4 Models and inference environment

We evaluate Gemini 2.5 Pro, Flash, and Qwen2.5-Omni7B. Gemini runs use the `google.genai` SDK. For Qwen, we mirrored the same pipeline on NYU's HPC with provider-specific chat/message shims, but identical prompts, decoding settings, and evaluation. Runs are deterministic (temp. $= 0$).

3 Results

Overall performance across modalities. Table 1 summarizes accuracy across models and prompting strategies. Performance depended strongly on modality and model. MIDI input yielded near-ceiling scores 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 deficits under LogicLM.

Modality differences. A robust modality gap was observed (Fig. 1A). Gemini models performed significantly better with MIDI ($p < .001$), though, this trend was less evident in Qwen. Syncopation Scoring and Chord Quality Identification showed the widest gaps for Gemini (MIDI $\approx$84–100% vs. audio $\approx$6–65%), confirming intact symbolic reasoning but weak audio perception. Transposition Detection was more robust, with smaller modality gaps.

ZS vs. FS. Collapsing across models and prompts, no significant main effects of shot were observed (Fig. 1B; all p's $> .05$). FS tended to help Syncopation in audio (e.g., Gemini Pro from $\sim$25% ZS to $\sim$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

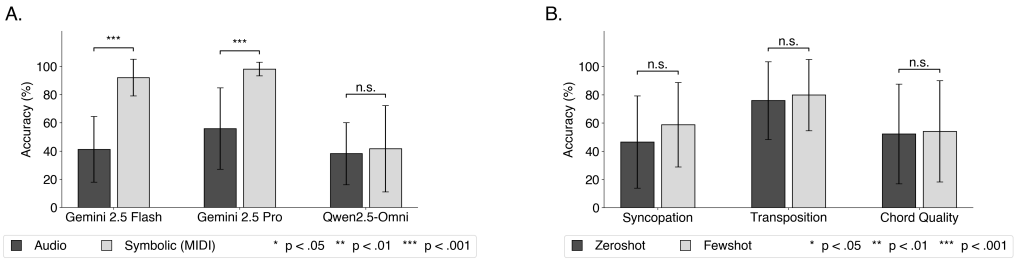

Figure 1: Model accuracy across modalities and prompting strategies. Error bars indicate standard deviation and significance is based on Welch's t-tests carried out with `SciPy`. Figure 1A shows the collapsed (Shot and Prompting Method) mean accuracy (%) for Gemini 2.5 Flash, Gemini 2.5 Pro, and Qwen 2.5-Omni across Audio (dark grey) and Symbolic (MIDI) (light grey) inputs. Both Gemini models performed significantly better on MIDI, and Qwen showed a similar trend. Figure 1B shows the collapsed (Model and Prompting Method) overall model accuracy by task (Syncopation, Transposition, Chord Quality) under Zeroshot (dark grey) and Fewshot (light grey) prompting. No significant effects of Shot were exhibited.

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.

4 Discussion

Our findings converge on a simple but consequential claim: multimodal LLMs reason effectively over symbolic music data, yet still fail to "listen" reliably. 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. FS examples helped only where perceptual calibration mattered (e.g., rhythmic counting); neither CoT nor LogicLM compensated for upstream hearing errors.

This gap matters because people experience music through audio, not symbolic proxies. A claim of "musical understanding" requires that models handle tracks directly, as one would text or video. Symbolic formats strip away the features making music meaningful (micro-timing, articulation, expressive nuance) so ceiling performance on MIDI should not be mistaken for audio-native competence.

Closer inspection shows that apparent successes can reflect superficial heuristics rather than genuine listening. In Transposition Detection, for instance, Gemini Pro often preserved sequence length while failing to capture intervallic structure and contour; LogicLM exposed these degenerate strategies by enforcing musical consistency, whereas Standalone/CoT could mask such fundamental errors. A similar dynamic appears in Chord ID (audio), where confusions between nearby qualities (e.g., major vs. dominant) and voicing/inversion artifacts lead to mid-level accuracy even without the schema burden of LogicLM. Table 2 illustrates this failure mode.

Table 2: Sample transposition failures with different contours (↓↑↑↓... vs ↑↓↓↑...).

	E♭ Major	**G Major**
Gemini Pro	`[55, 51, 55, 58, 51, 55, ...]`	`[59, 55, 59, 62, 55, 59, ...]`
Ground Truth	`[67, 72, 67, 65, 67, 70, ...]`	`[71, 76, 71, 69, 71, 74, ...]`

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. 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 on small perceptual errors. LLMs that acquire genuine understanding could also be music education [28] and user-centric music analysis tools [16, 17], enabling interactive systems that can teach musical structure and foster deeper engagement with personal music listening.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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*, 2025.

[5] 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.

[6] 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.

[7] 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)*, 2025.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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 2025.

[13] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th International Conference on World Wide Web*, page 285–295, 2001.

[14] K. Chen, Beici Liang, Xiaoshuang Ma, and Minwei Gu. Learning audio embeddings with user listening data for content-based music recommendation. *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2020.

[15] Hope Mogale and M.B Esiefarienrhe. Optimizing recommendation algorithms using self-similarity matrices for music streaming services. In *International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems*, pages 1–4, 2021.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

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

[23] Diana Deutsch. Octave generalization and tune recognition. *Perception & Psychophysics*, 11 (6):411–412, 1972.

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

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

[26] 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.

[27] 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*, 2025.

[28] Lingxi Jin, Baicheng Lin, Mengze Hong, Hyo-Jeong So, and Kun Zhang. Learning by teaching: Enhancing music learning through llm-based teachable agents. In *International Conference on Artificial Intelligence in Education*, pages 148–155. Springer, 2025.

A Appendix: `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 3 below.

Table 3: 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

B Appendix: `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 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"

C Appendix: 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 LOGIC-LM self-refinement idea of using solver feedback to repair the symbolic form.

C.1 solver.py

```python
# 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}

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
```