Enhancing Reasoning Efficiency and Domain Adaptability of Server-Linked LLMs by Combining with a Customized Deep Learning Model

Junbum Cho

Received May 12, 2025
Accepted August 11, 2025
Electronic access August 30, 2025

Generative AIs capabilities rely on neural networks pre-trained on vast datasets, but this approach faces challenges from data irregularities and high computational costs up to millions of dollars. Even after extensive training, task contamination from overlapping training data often compromises few-shot learning evaluations, creating false impressions of Large Language Models (LLMs) true capabilities. This research introduces the DL-LLM Modular System, a novel method that enhances server-linked language models on-device performance in audio classification by integrating a specialized Deep Learning (DL) model with LLM through a modular architecture. The modular system divides complex perceptual processing and reasoning tasks involving audio data between specialized DL models and LLMs. For this research, a simple DL model using TensorFlow and Teachable Machine architecture was developed to classify five distinct audio types and integrate with an LLM. Compared to baseline LLMs without DL integration, this approach demonstrates significant improvements in audio-related reasoning efficiency and accuracy by allowing LLM to focus solely on data interpretation and reasoning. Experiment results show the system requires significantly fewer prompt shots for audio classification while reducing average end-to-end processing time from 23,979 ms to 18,824 ms for Gemini 2.5 Pro, a net improvement of 21.5%. Furthermore, the modular nature of the architecture enables easy and cost-effective customization to specific domain requirements and expanding potential applications across various fields involving audio classification tasks. Moreover, this architecture reduces deployment costs by running the DL model on-device, which lessens the computational load on the main server.

1 Introduction

1.1 Current Trend of LLM Training Methods and Their Implications

Up to early 2025, three advanced training methodologies define the landscape of Large Language Model (LLM) training. Prompt engineering has evolved into a sophisticated practice for guiding model outputs without modifying underlying architectures[1], with few-shot learning demonstrating particular promise for improving performance[2], though concern that it creates false impression also exists[3]. Retrieval Augmented Generation (RAG) has become a cornerstone of practical LLM implementation, with hybrid systems combining knowledge graphs and vector retrieval representing the cutting edge of this approach[4]. Efficient fine-tuning methods such as LoRA have gained prominence for their ability to adapt models with minimal computational resources, making customization more accessible[5].

Meanwhile, the economic landscape of LLM training reveals a staggering investment across the AI industry, with costs ranging from thousands to hundreds of millions of dollars, depending on model size and complexity. Frontier models like GPT-4 reportedly cost approximately $100 million to train[6–8], while mid-sized models like GPT-3 (175B parameters) require more than $4.6 million, and smaller 7B parameter models can be trained for around $30,000. These expenses are primarily driven by computational infrastructure, substantial energy consumption (GPT-3s training consumed 1,287 MWh of electricity[9], and the high salaries commanded by AI researchers and engineers. Cost-reduction strategies include efficient fine-tuning methods like LoRA[10], model architecture optimization through techniques such as mixed-precision training and knowledge distillation[11,12], and infrastructure optimization via strategic cloud resource management[13].

1.2 Research Objective

Research prior to March 2025 had relied on generic multimodal integration[14], efficiency through existing architecture optimization[15], Reinforce Learning Human Feedback (RLHF)-based fine-tuning[16], and standard pre-training or fine-tuning pipelines[17]. These approaches face challenges from data irregularities and computational costs reaching millions of dollars. This paper takes the first step toward improving server-linked on-device LLMs reasoning efficiency and a cost-effective server-adaptability through its combination with a DL model, primarily focusing on audio processing, aiming to distinguish five categories - Background Noise, Screaming Noise, Gunshot, Glass Shattering, and Siren.

This paper introduces the DL-LLM Modular System, a novel framework that enhances LLM reasoning efficiency, decision latency, and computational cost through the strategic integration of specialized DL models with particular emphasis on complex audio data analysis. The modular systems performance was systematically evaluated across multiple frontier LLMs (OpenAI ChatGPT, Google Gemini, DeepSeek, and Anthropic Claude) to quantify improvements in all three areas compared to traditional approaches. Fig 1 represents the operating mechanism of the

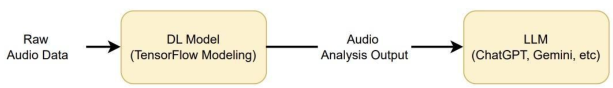


Fig. 1 Simplified Operation Diagram of the DL-LLM Modular System.

DL-LLM Modular System. The raw audio file is accepted by the DL model trained based on the TensorFlow modeling method, whose analysis outputs are directly fed to the LLM in a text format.

1.3 Related Works

Prior work in modular or hybrid architectures combining deep learning with large language models has demonstrated promising adaptability in few-shot scenarios. Notably, Flamingo[18] integrates pretrained vision and language models using gated cross-attention mechanisms, enabling rapid adaptation to multimodal tasks such as visual question answering with minimal fine-tuning.

Unlike Flamingo, which fuses multimodal inputs in a shared transformer pipeline, our DL-LLM Modular System distinctly separates perceptual processing (via a specialized audio DL model) from reasoning (via an LLM), facilitating lower computational cost and enhanced task-specific interpretability. This architectural modularity allows greater flexibility in customizing domain-specific inputs while maintaining general reasoning capabilities and efficiency.

2 Results

2.1 Overview

The evaluation of the DL-LLM Modular System was conducted through two complementary testing branches. The first testing branch assessed the system's fundamental audio classification capabilities and reasoning efficiency by comparing the sound classification results and number of required shots the against baseline server-linked LLMs without specialized DL model integration. The second test extended beyond basic classification to evaluate inference capabilities and reasoning efficiency with latency measurements, challenging both systems with sophisticated tasks while conducting comprehensive performance evaluation including processing time analysis.

Both tests were completed on identical hardware configurations: a MacBook Pro (14-inch, 2021) equipped with an Apple M1 Pro chip featuring a 10-core CPU (8 performance cores + 2 efficiency cores), 14-core GPU, 16-core Neural Engine, and 32GB unified LPDDR5 memory running macOS 15.2 (24C101). However, different software platforms as detailed in their respective methodology sections.

2.2 Evaluating Modular System Performance Against ChatGPT-4o in Audio Classification and Reasoning Tasks

2.2.1 Overview

The analysis results and number of shot attempts taken between the DL-LLM Modular System and a baseline model (raw GPT-4o) were compared. Conducted with a non-inference model, this test strictly assessed the modular system and the raw LLMs recognition accuracy and efficiency for four distinct acoustic events: Screaming, Glass Shattering, Gunshot, and Siren. To do so, each models response and the number of shots until reaching the correct recognition were counted. Shot attempts refer to the number of example instances provided to the model before classification: Zero Shot (no examples), One Shot (one example), and Two Shot (two examples), with higher shot counts indicating the model required more contextual examples to achieve accurate recognition. For the testing, audio files illustrated in the Table 1 were utilized, where total eight test sessions were done using all the eight prepared audio files were utilized. Some of the results are illustrated in the following sections.

2.2.2 Challenges with Existing Multimodal LLMs

While seeking appropriate benchmarks for audio classification tasks, several existing multimodal LLMs were evaluated but revealed significant limitations that precluded meaningful comparison.

Although OpenAI's Whisper-Integrated GPT conducts audio processing using a separate model, Whisper, this system proves inadequate for the required comparison needs. Whisper primarily focuses on transcribing human speech to text and cannot effectively interpret brief non-verbal audio segments, such as the 1-second audio files required for these experiments.

Microsoft's Project Rumi presented similar limitations. Beyond being restricted to research access only, this system focuses on enhancing LLM interpretation depth by separating vision and audio models to extract sentiment and emotional cues from user input. This approach does not align with the objective of classifying discrete audio types.

Table 1 Details of Audio Files Used for DL-LLM Modular System Testing for Section 2.1. Duration and description of each test audio file is illustrated by the table.

Audio Data	Length (sec)	Audio File Information	File Count	Detail of the Sound
Screaming	1	WAV file format 16 kHz sampling rate	1	The test utilized crowd screaming audio featuring persistent, intense shouting.
		16-bit PCM (Pulse Code Modulation) encoding Mono channel configuration	1	The test utilized female screaming audio featuring short, abrupt shouting.
Siren	1	WAV file format 16 kHz sampling rate	1	The Siren sound had a repeating, oscillating pattern typical of police sirens and alternating frequency bands, suggesting the classic wee-woo pattern
		16-bit PCM encoding Mono channel configuration	1	The Siren sound featured a continuous rising and falling pitch in long sweeps, creating a warbling, undulating pattern typical of air raid or tornado warning sirens, distinct from the rapid alternating tones of emergency vehicles.
Glass Shattering	1	WAV file format 16 kHz sampling rate	1	The glass shattering sound featured a breaking glass cup, with the test focusing on the peak impact moment that produces the most characteristic shattering pattern.
		16-bit PCM encoding Mono channel configuration	1	The glass shattering sound featured a large window pane breaking, with the test focusing on the cascading fragments creating multiple overlapping crash sequences rather than a single sharp impact.
Gunshot	1	WAV file format 16 kHz sampling rate	1	The gunshot audio featured a sharp, distinctive Tang firing sound typical of small arms discharge.
		16-bit PCM encoding Mono channel configuration	1	The gunshot audio featured a deep, booming rifle discharge with extended reverberation, creating a thunderous echo rather than a sharp crack.

Total eight audio files detailed in Table 1 were used for the following experimentations in setion 2.2 and 2.3, where the first of the two audio file was containing a similar sound with the audio file used for the training while the other, second file, had a variation from the training dataset. The detailed audio data description used for the training can be seen at table 12 located at section 4.3, the Methods section.

2.2.3 Challenges through Direct Conversation with LLM

Given that ChatGPT-4o natively processes multiple input and output modalities, including audio, in a web environment (accessible through chatgpt.com), a direct approach was attempted by playing the desired audio files directly to the model, exploting GPT-4os conversational system. This method proved unsuccessful, reinforcing that language models alone have inherent limitations accurately classifying audio signals, particularly non-verbal audio content.

Table 2 — Experimental Conversation Testing ChatGPT-4o's Direct Audio Processing Capabilities. Transcript of experimental conversation testing ChatGPT-4o's real-time audio processing capabilities. "Time" indicates researcher's audio input timestamps (ChatGPT responses are text-based). Results demonstrate limitations in processing environmental audio versus direct file uploads.

2.2.4 Approach

Sound classification accuracy and the number of shot examples required to achieve correct classification were quantified using a web-based GPT interface, which allowed for iterative testing and precise shot counting through direct response evaluation. The baseline model received raw audio inputs paired with prompts, while the DL-LLM system processed pre-classified outputs from the DL model. This methodology enabled precise tracking of response accuracy and facilitated an iterative questioning protocol for shot-attempt measurement. For both models, audio files matching the DL models input specifications were prepared, as illustrated in Table 1. All files were renamed to "unknown_audio_file.wav" to eliminate filename-induced bias.

Table 2 Experimental Conversation Testing ChatGPT-4o's Direct Audio Processing Capabilities. Transcript of experimental conversation testing ChatGPT-4o's real-time audio processing capabilities. "Time" indicates researcher's audio input timestamps (ChatGPT responses are text-based). Results demonstrate limitations in processing environmental audio versus direct file uploads.

Speaker	**Transcript**
Researcher	Hello, GPT.
ChatGPT	Hey there! How's it going? What's on your mind today?
Researcher	Can you distinguish the sound that I'm playing through my cell phone?
ChatGPT	I can't directly hear or identify sounds from your environment, but if you describe it to me or let me know what you're trying to figure out, I'll do my best to help!
Researcher	If I play the sound that I want through my cell phone this time, can you distinguish it?
ChatGPT	I can't actually pick up or analyze audio directly from your surroundings. But if you tell me what kind of sound it is or what you're curious about, I can still try to help out!
Researcher	So, to be exact, you mean that you cannot hear the sound in spite of that I'm playing it through my cell phone?
ChatGPT	Exactly! I can't hear any real-time audio from your environment. But if you have any questions or need help with something related to the sound, feel free to let me know!
Researcher	What happens if I give you the audio file directly?
ChatGPT	If you share the audio file with me directly, I can definitely help analyze it or provide some insights based on whats in the audio. Feel free to upload it whenever you'd like!

2.2.5 Data Collected

- Screaming Sound
  The first test evaluated the screaming sound of people, truncated into 1 second. The table below shows results for audio files similar to those used in training.

Table 3 Comparative Analysis of Sound Identification for Screaming Sound: Raw GPT-4o vs DL-LLM Modular System. '-' symbols indicate cases where the GPT model successfully classified the audio input on the first attempt, requiring no additional trials.

Judgment Input & Output		**GPT 4o (Recognition) - GPT 4o (Judgement)**	**DL (Recognition) - GPT 4o (Judgement) Modular System**
Zero Shot	**Input 1**	Can you identify what sound this is? (Unknown_Audio_File.wav attached.)	Can you identify what sound this is? The current detected class: - Background Noise: 0.0005 - Glass Shattering: 0.0000 - Gunshot: 0.0018 - Screaming: 0.9534 - Siren: 0.0443
	Output 1	*(Generates a frequency spectrum)* Here's the frequency spectrum of the audio file. (...) To get a better idea of what it is, would you like me to: Visualize it as a spectrogram?, Try classifying it using a pre-trained sound recognition model?, Or do you already have a few guesses you want to confirm?	The dominant classification is screaming, which makes it the most probable sound.
One Shot	**Input 2**	Please proceed with your plan to determine what sound this audio file contains.	-
	Output 2	We're running into limitations here the necessary library for YAMNet (tensorflow) isnt currently available in this environment.	-
Two Shot	**Input 3**	Please simply proceed with your own plan to make a final decision on what sound this audio file contains.	-
	Output 3	A microwave or oven timer alert A digital watch alarm A sensor or equipment status beep	-

As seen, the baseline GPT-4o model incorrectly inferred the given sound signal as microwave or oven timer alert sounds despite multiple shot attempts. In contrast, the DL-LLM Modular System successfully identified the correct sound-based DL models classification outputs with a single-shot attempt.

- Glass Shattering Sound
  The second test evaluated a 1-second glass-shattering sound. The table below shows results for audio files similar to those used in training.

Table 4 Comparative Analysis of Sound Identification for Glass Shattering: Raw GPT-4o vs DL-LLM Modular System. '-' symbols indicate cases where the GPT model successfully classified the audio input on the first attempt, requiring no additional trials.

Judgment Input & Output		**GPT 4o (Recognition) - GPT 4o (Judgement)**	**DL-LLM Modular System**
Zero Shot	**Input 1**	Can you identify what sound this is? (Unknown_Audio_File_2.wav attached.)	Can you identify what sound this is? The current detected class: - Background Noise: 0.0005 - Glass Shattering: 0.9995 - Gunshot: 0.0000 - Screaming: 0.0000 - Siren: 0.0000
	Output 1	*(Generates a spectrogram and ceases the analysis.)* To identify the sound more precisely, I can: 1. Play the sound back for you. 2. Try identifying the type (e.g., beep, voice, mechanical noise, etc.). 3. Extract features like pitch or formants.	Based on those detection probabilities, the sound is almost certainly glass shattering.
One Shot	**Input 2**	Please proceed with your plan to determine what sound this audio file contains.	-
	Output 2	- Electronic beeps - Alarm signals - Bird chirps - Certain mechanical/electrical sounds	-

As seen, the baseline GPT-4o model incorrectly inferred the given sound signal as a "synthetic electronic alarm tone" sound despite multiple shot attempts. while the DL-LLM Modular System successfully identified the correct sound with a single-shot attempt.

- Siren Sound
  The continued testing evaluated the 1-second Siren sound. The table below shows results for audio files similar to those used in training.

Table 5 Comparative Analysis of Sound Identification for Siren: Raw GPT-4o vs DL-LLM Modular System. '-' symbols indicate cases where the GPT model successfully classified the audio input on the first attempt, requiring no additional trials.

Inputs & Responses		**Raw ChatGPT-4o**	**DL-LLM Modular System**
Zero Shot	**Prompt 1**	Can you identify what sound this is? (Unknown_Audio_File_3.wav attached.)	Can you identify what sound this is? The current detected class: - Background Noise: 0.0000 - Glass Shattering: 0.0000 - Gunshot: 0.0000 - Screaming: 0.0002 - Siren: 0.9998
	Response 1	*(Generates a spectrogram and ceases the analysis.)* If you can tell me more about the contextwhere the sound came from or what you suspect it might beI can give you a more specific analysis or try to match it more closely.	Based on the detection results you shared, the sound is most likely a siren.
One Shot	**Prompt 2**	Please proceed with your plan to determine what sound this audio file contains.	-
	Response 2	A short electronic beep or alert tone.	-

The observed trend in the two previous test sessions continues.

- Gunshot Sound
  The final testing evaluated the two model architectures ability to discern gunshot sounds. The table below shows results for audio files having variations compared to those used in training.

Table 6 Comparative Analysis of Sound Identification for the Gunshot Sound: Raw GPT-4o vs DL-LLM Modular System. '-' symbols indicate cases where the GPT model successfully classified the audio input on the first attempt, requiring no additional trials.

Judgment Input & Output		**GPT 4o (Recognition) - GPT 4o (Judgement)**	**DL-LLM Modular System**
Zero Shot	**Prompt 1**	Can you identify what sound this is? (Unknown_Audio_File_4.wav attached.)	Can you identify what sound this is? The current detected class: - Background Noise: 0.0218 - Screaming: 0.1687 - Gunshot: 0.7475 - Siren: 0.0446 - Glass Shattering: 0.0174
	Response 1	*(Generates a waveform and spectrogram and ceases the analysis.)* Heres the spectrogram of the audio you uploaded. It shows how the frequency content of the sound evolves over time. To identify the sound, I need to analyze a few characteristics: - Duration - Frequency range - Temporal pattern (e.g., repeating, constant tone, sudden bursts)	Based on the detection scores you gave, the sound is most likely a gunshot.
One Shot	**Prompt 2**	Please proceed with your plan to determine what sound this audio file contains.	-
	Response 2	Most likely source: A steady electronic tone such as a test signal, alarm tone, or electronic device alert.	-

To obtain statistically robust datasets, four additional tests were conducted, using the remaining audio file illustrated in Table 1. The entire results are presented in Table 7 below.

2.3 Performance Evaluation of the Modular System Employing Various Inference LLM Models

2.3.1 Overview

Beyond shot attempts and classification accuracy, to evaluate how the modular system affects LLM reasoning efficiency in terms of latency, tests were expanded beyond the initial ChatGPT-4o implementation to include four additional inference LLMs: Gemini 2.5 Pro, Claude 3.7 Sonnet, DeepSeek R1, and Grok-3. Although Metas Llama 4 Maverick and OpenAIs ChatGPT-o1-pro were initially targeted for testing, both were excluded due to limited publicly available APIs. Thus, the evaluation framework specifically assessed the available models advanced reasoning abilities beyond basic audio signal analysis through targeted reasoning tasks.

2.3.2 Approach

Given the growing prominence of inference language models[19], this test assessed the LLMs reasoning capability through temporal efficiency in audio classification tasks, where the processing latency the time required by different architectures to distinguish between acoustic event categories reliably was evaluated. The experimental framework was created through the modular architecture developed in Python using Google Colabs computational environment, illustrated in the following system diagrams.

Using the audio files in Table 7, each sound category under-

Table 7 Comparative Prompt Shot Counts for Audio Classification: Baseline GPT-4o vs. DL-LLM Modular System. The number of prompt attempts required to correctly classify four acoustic event typesScreaming, Glass Shattering, Siren, and Gunshotacross two independent test sessions per sound are shown. The DL-LLM Modular System consistently reduces the number of required shots by one (median reduction = 1), a difference that is statistically significant by a one-sided Wilcoxon signed-rank test (W = 36, p = 0.0039).

Sound Type	Test Sessions	Baseline GPT-4o	DL-LLM Modular System	Difference (Base-lineModular Structure)
Screaming	**Test 1**	3	1	2
	Test 2	3	1	2
Glass Shattering	**Test 1**	2	1	1
	Test 2	3	1	2
Siren	**Test 1**	2	1	1
	Test 2	2	1	1
Gunshot	**Test 1**	2	1	1
	Test 2	2	1	1

With eight paired observations, two for each sound type, the one-sided Wilcoxon signed-rank test yields W = 36, p = 0.003906, and a median difference of 1 shot (Baseline Modular). Because p ¡ 0.05, it can be concluded that the DL-LLM Modular System meaningfully requires fewer prompt shots than raw GPT-4o. In every case, the integrated DL-preprocessing pipeline reduced the required shots by one. Furthermore, as shown in Figures 2 to 5, the modular system achieved higher classification accuracy than raw GPT-4o by correctly identifying sound types. This demonstrates consistent improvement in the LLM's reasoning efficiency when interpreting audio signals, effectively addressing the LLM's inherent limitations in processing acoustic data without specialized preprocessing.

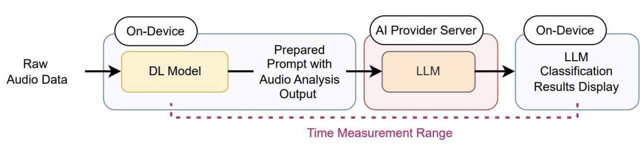


Fig. 2 DL-LLM Modular System Architecture with 7 Measurement Settings.

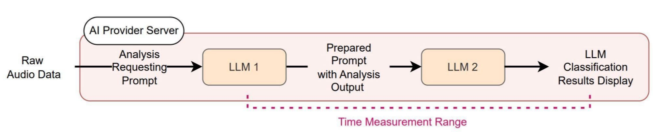


Fig. 3 Baseline LLM Architecture with Response Time Measurement Settings.

went five test sessions: the first two used sounds with slight variations from the training dataset, while the remaining three used sounds similar to those in the training data.

2.3.3 Limitations

All latest LLMs planned for testing, excluding Gemini 2.5 Pro, did not accept audio files. Those who did not accept audio file analysis were excluded from the experiment. Ultimately, Gemini 2.5 Pro became the only model that was tested. Furthermore, since API performance metrics are susceptible to server traffic fluctuations, this may have introduced minor measurement inconsistencies.

2.3.4 Data Collected

Table 8 Performance Comparison of the Baseline models and the DL-LLM Modular System for Gemini 2.5 Pro and Deepseek R1. '-' symbols indicate cases where the specified architecture (Baseline Model) were non-functional as the raw LLM API was incapable of processing raw audio files.

			Gemini 2.5 Pro		**DeepSeek-R1**	
			Baseline Model (Gemini 2.5 Pro)	**DL-LLM Modular System**	**Baseline Model (DeepSeek-R1)**	**DL-LLM Modular System**
			Time Spent (ms)	**Time Spent (ms)**	**Time Spent (ms)**	**Time Spent (ms)**
Sound Types	**Test Sessions**	**Processing Time**				
Screaming	**1**	**LLM / DL**	7655.62	5.58	-	10.98
		LLM	15348.03	18983.09	-	25560.9
		Total	23003.65	18988.67	-	25571.88
	2	**DL Model**	5128.9	26.9	-	12.89
		LLM	16360.7	17110.74	-	34369.59
		Total	21489.6	17137.64	-	34382.48
	3	**DL Model**	6060.24	8.47	-	6.1
		LLM	20430.29	18760.5	-	36974.56
		Total	26490.52	18768.97	-	36980.66
	4	**DL Model**	5633.03	17.72	-	5.25
		LLM	15086.56	21504.29	-	42999.72
		Total	20719.59	21521.71	-	43004.97
	5	**DL Model**	6512.43	18.87	-	6.93
		LLM	16582.59	18336.07	-	29368.06
		Total	23095.02	18354.94	-	29374.99
Siren	**1**	**LLM / DL**	5644.49	5.61	-	17.88
		LLM	14356.95	17544.13	-	37348.95
		Total	20001.44	17549.74	-	37366.83
	2	**LLM / DL**	6162.28	8.53	-	5.33
		LLM	18611.46	18758.62	-	37140.01
		Total	24773.74	18767.15	-	37145.34
	3	**LLM / DL**	6923.01	5.46	-	9.44
		LLM	20860.37	15907.18	-	56452.92
		Total	27783.38	15912.64	-	56462.36
	4	**LLM / DL**	4214.63	5.82	-	14.46
		LLM	11713.67	16923.53	-	29897.9
		Total	15928.3	16929.35	-	29912.36
	5	**LLM / DL**	5243.74	5.24	-	5.77
		LLM	17977.47	16019.2	-	35549.17
		Total	23221.21	16024.44	-	35554.94
Glass Shattering	**1**	**LLM / DL**	5654.29	9.95	-	8.88
		LLM	23548.87	18864.91	-	36994.42
		Total	29203.16	18874.86	-	37003.3
	2	**LLM / DL**	5409.94	13.29	-	8.18
		LLM	19055.01	19818.08	-	33336.72

		Total	24464.95	19831.37	-	33344.9
	3	**LLM / DL**	6518.66	4.63	-	6.55
		LLM	19124.91	19296.12	-	31461.11
		Total	25643.57	19300.75	-	31467.66
	4	**LLM / DL**	5586.25	5.68	-	5.4
		LLM	15877.37	18032.3	-	29161.41
		Total	21463.62	18037.98	-	29166.81
	5	**LLM / DL**	5995.05	4.78	-	6.41
		LLM	21702.44	17232.83	-	32121.41
		Total	27697.49	17237.61	-	32127.82
Gunshot	**1**	**LLM / DL**	8002.99	11.27	-	10.15
		LLM	18183.88	20154.45	-	45523.59
		Total	26186.87	20165.72	-	45533.74
	2	**LLM / DL**	4788.78	4.87	-	5.99
		LLM	16580.79	20354.06	-	42679.86
		Total	21369.57	20358.93	-	42685.85
	3	**LLM / DL**	6081.64	4.8	-	6.28
		LLM	20845.39	19857.54	-	36940.83
		Total	26927.03	19862.34	-	36947.11
	4	**LLM / DL**	5971.72	11.77	-	6.14
		LLM	20644.07	21898.42	-	34303.28
		Total	26615.79	21910.19	-	34309.42
	5	**LLM / DL**	6934.85	14.62	-	5.35
		LLM	16573.38	20925.55	-	32120.14
		Total	23508.23	20940.17	-	32125.49

The test continued with Comparison of the DL-LLM Modular System and the baseline model utilizing Grok-3 and Claude 3.7 Sonnet where as noted, the modular system could not be attained as the APIs did not support direct audio injection.

Table 9 Performance Comparison of the Baseline models and the DL-LLM Modular System for Grok 3 and Claude 3.7 Sonnet. '-' symbols indicate cases where the specified architecture (Baseline Model) were non-functional as the raw LLM API was incapable of processing raw audio files.

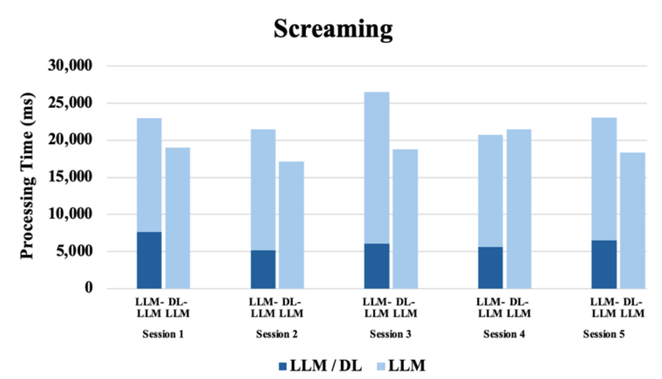
			Grok 3		**Claude 3.7 Sonnet**	
			Baseline Model	DL-LLM Modular System	Baseline Model	DL-LLM Modular System
			Time Spent (ms)	Time Spent (ms)	Time Spent (ms)	Time Spent (ms)
Sound Types	**Test Sessions**	**Processing Time**	Time Spent (sec)	Time Spent (sec)	Time Spent (sec)	Time Spent (sec)
Screaming	**1**	**LLM / DL**	-	12.77	-	10.39
		LLM	-	33510.09	-	6724.73
		Total	-	33522.86	-	6735.12
	2	**LLM / DL**	-	8.01	-	6.21
		LLM	-	13735.82	-	4571.55
		Total	-	13743.83	-	4577.76
	3	**LLM / DL**	-	6	-	6.14
		LLM	-	7500.7	-	5911.23
		Total	-	7506.7	-	5917.37
	4	**LLM / DL**	-	6.19	-	7.07

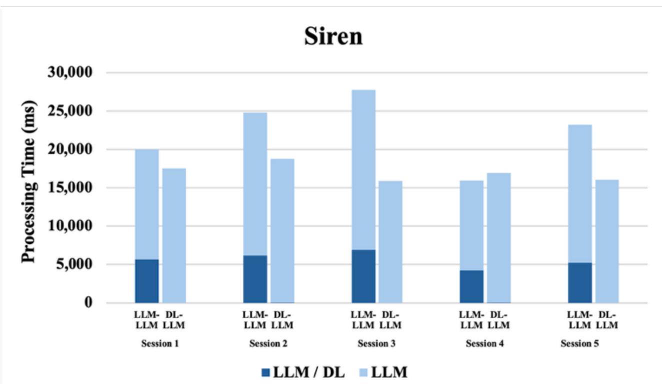
		LLM	-	7303.8	-	5452.55
		Total	-	7309.99	-	5459.62
	5	**LLM / DL**	-	5.17	-	5.24
		LLM	-	26386.29	-	5371.34
		Total	-	26391.46	-	5376.58
Siren	**1**	**LLM / DL**	-	5.62	-	5.2
		LLM	-	9802.03	-	7141.74
		Total	-	9807.65	-	7146.94
	2	**LLM / DL**	-	7.92	-	6.1
		LLM	-	14303.74	-	6287.14
		Total	-	14311.66	-	6293.24
	3	**LLM / DL**	-	5.39	-	8.83
		LLM	-	16448.47	-	5921.57
		Total	-	16453.86	-	5930.4
	4	**LLM / DL**	-	5.43	-	5.03
		LLM	-	11594.53	-	5470.2
		Total	-	11599.96	-	5475.23
	5	**LLM / DL**	-	6.84	-	7.98
		LLM	-	16977.67	-	5515.24
		Total	-	16984.51	-	5523.22
Glass Shattering	**1**	**LLM / DL**	-	34.84	-	6.44
		LLM	-	26409.05	-	5190.28
		Total	-	26443.89	-	5196.72
	2	**LLM / DL**	-	11.69	-	5.18
		LLM	-	34014.35	-	5346.21
		Total	-	34026.04	-	5351.39
	3	**LLM / DL**	-	6.48	-	8.51
		LLM	-	8331.06	-	4373.23
		Total	-	8337.54	-	4381.74
	4	**LLM / DL**	-	5.47	-	5.48
		LLM	-	14209.06	-	5341.25
		Total	-	14214.53	-	5346.73
	5	**LLM / DL**	-	8.12	-	10.15
		LLM	-	13791	-	4812.23
		Total	-	13799.12	-	4822.38
Gunshot	**1**	**LLM / DL**	-	6.26	-	5.23
		LLM	-	14319.17	-	6471.34
		Total	-	14325.43	-	6476.57
	2	**LLM / DL**	-	6	-	5.28
		LLM	-	21331.89	-	5629.15
		Total	-	21337.89	-	5634.43
	3	**LLM / DL**	-	6.31	-	5.25
		LLM	-	17470.81	-	6532.99
		Total	-	17477.12	-	6538.24
	4	**LLM / DL**	-	5.21	-	8.17
		LLM	-	15544.24	-	5939.25
		Total	-	15549.45	-	5947.42
	5	**LLM / DL**	-	8.12	-	5.19
		LLM	-	22035.16	-	4761.8
		Total	-	22043.28	-	4766.99

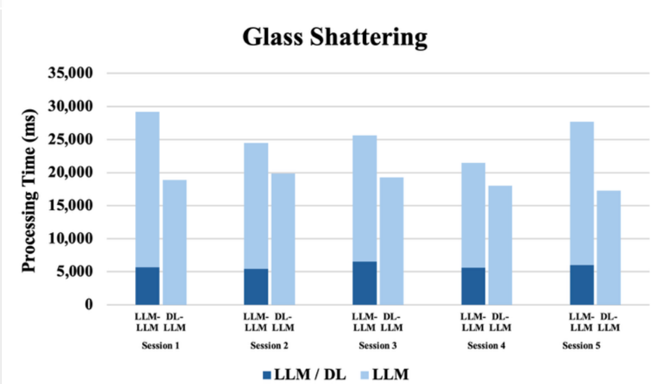
Since Gemini 2.5 Pro was the only model that could accept the raw audio file and thus have the comparative dataset with the baseline model, after the test, the average processing time duration between the baseline and DL-LLM Modular architecture using Gemini 2.5 Pro was calculated. The metrics were derived by aggregating 40 sound classification results 20 trials per Gemini-based model architecture then dividing each dataset by its respective trial count (20). Across all four sound categories, the DL-LLM Modular System reduced average end-to-end processing time via the Gemini 2.5 Pro API from 23,979 ms to 18,824 msa net improvement of approximately 5,155 ms ($\sim$21.5% faster) . This reduction reflects both the modular division of labor and API response variability.

Table 10 Average Model Architecture Processing Time (in milliseconds) of the Baseline models and the DL-LLM Modular System.

Model	Baseline Gemini 2.5 Pro	DL-LLM Modular System Gemini 2.5 Pro
Avg Processing Time (ms)	23979.34	18823.76







For a more reliable analysis, a series of paired t-tests comparing baseline inference times to those of the DL-LLM Modular System for Gemini 2.5 Pro were conducted. In essence, they demonstrate that integrating the DL model significantly

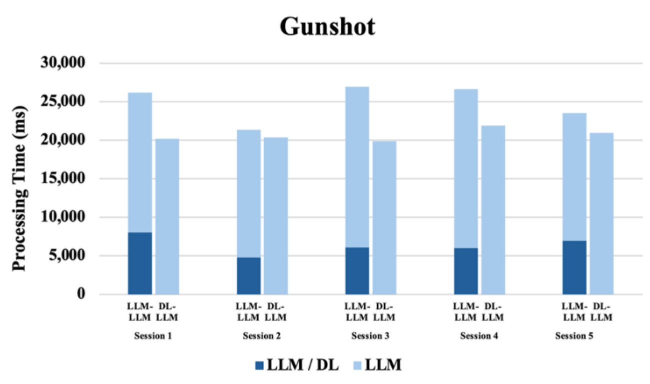


Fig. 4 Processing Time Comparison with the Baseline and the DL-LLM Modular System Utilizing Gemini-2.5-Pro.

improves the LLMs ability to interpret audio signals (i.e., it reduces the time required for reliable classification). Although Gemini incorporates native audio processing via a sophisticated tokenization technique transforming acoustic waveforms into discrete tokens for its neural architecture [20] this built-in multimodal pipeline proved less efficient. For example, on Glass Shattering trials, the modular system achieved a mean reduction of 7,038 ms (t(4)=4.865, p=0.008), and on Gunshot trials it saved 4,274 ms on average (t(4)=3.855, p=0.018). Even for Screaming, the average gain was 4,005 ms (t(4)=2.922, p=0.043). These p-values all fall below $\alpha = 0.05$, confirming that the observed speedups are unlikely to occur by chance. In contrast, the Siren condition showed a smaller 5,305 ms improvement that did not reach statistical significance (t(4)=2.433, p=0.072), which may be attributed to variability in server traffic rather than inherent model differences, since both the baseline and DL-LLM Modular System evaluations relied on the same API infrastructure. Overall, the statistically significant reductions in processing time demonstrate that the modular design meaningfully enhances reasoning efficiency and inference latency in LLMs. Results from both experiments from sections 2.2 and 2.3 indicate that integrating the LLM with the DL model significantly improves domain adaptability for specialized audio tasks, as elaborated in the conclusion section.

3 Conclusion

3.1 Overview

To build the modular architecture and measure latency improvements, a lightweight CNN was trained using Google Teachable Machine and exported as a TensorFlow Lite model to recognize five 1-second sounds (background noise, screaming, glass shattering, gunshot, and siren). A Python/Colab ”Module 1” standardized each audio clip. ”Module 2” then to performed the on-device inference using the DL model and transmitted the probability vector to inference-only LLM APIs (Gemini 2.5 Pro, DeepSeek R1, Claude 3.7 Sonnet, and Grok-3) while a high-precision timer recorded end-to-end processing time. Results were compared against a Baseline LLM-LLM System that provided raw audio files directly to the LLM API without modular preprocessing.

To assess sound classification accuracy, two ChatGPT-4o models were tested: one receiving raw audio files and another receiving Teachable Machine classification results. The model provided with classification results demonstrated superior reasoning efficiency compared to the raw audio model.

Across eight paired tests spanning four sound types (Screaming, Glass Shattering, Siren, Gunshot), the DL-LLM Modular System consistently reduced the number of prompt attempts by one compared to raw GPT-4o (median difference = 1 shot; onesided Wilcoxon signedrank W = 36, p = 0.003906). In latency evaluations using Gemini 2.5 Pro, average end-to-end processing time fell from 23 979.34 ms to 18 823.76 ms ($\Delta \approx 5155.58 ms$). Paired t-tests confirmed significant speedups for Glass Shattering (Δ = 7 038 ms, t(4)=4.865, p=0.008), Gunshot (Δ = 4 274 ms, t(4)=3.855, p=0.018), and Screaming (Δ = 4 005 ms, t(4)=2.922, p=0.043), while the Siren condition showed a smaller, nonsignificant improvement of 5 305 ms (p=0.072).

A key limitation of the experiments is that the end-to-end latency measurements in Section 4.2 depend on the Gemini 2.5 Pro API. API response times can vary due to network conditions and server load. This limitation was attempted to be resolved by repeating the latency measurement multiple times. Additionally, other inference LLM APIs do not support audio input, which prevented the establishment of alternative DL-LLM Modular Systems and limited testing to the Baseline LLM-LLM structure. Consequently, the latency improvement analysis was constrained to results from the Gemini 2.5 Pro API. Below illustrates a few important insights that can be gained from the DL-LLM Modular Structure and its experiment results.

3.2 Insights Gained

- Enhanced Reasoning Efficiency The DL-LLM Modular System implements a strategic division of labor between complementary AI components, where a specialized DL model handles complex perceptual data processing (audio analysis) before transferring these structured outputs to the LLM. This architecture partially liberates the language model from complicated processing constraints, allowing it to focus more effectively on high-level reasoning and interpretation. As demonstrated in Section 2.2, this specialization significantly improved classification accuracy while requiring fewer shot attempts compared to conventional approaches-concrete evidence of enhanced reasoning efficiency through targeted task allocation.

- Reduced Server Dependency and Response Latency It is well known that LLMs server-dependency leads to increased response latency[21]. This division of labor makes LLM functionality more efficient by reducing the computational burden on servers and minimizing performance latency. Section 2.2 demonstrates how these reduced server operations lead to faster response generation. This approach provides an efficient way to run LLM methods, achieving higher response accuracy while simultaneously reducing server load and response times.

- Cost-Effective and Potential Domain Adaptation Moving on, based on the DL models inherent customizability, dataset flexibility, and relatively low training costs, the DL-LLM Modular System allows active utilization in various fields involving audio tasks at a significantly lower cost than traditional LLM training. In data-limited environments, the DL model component effectively compensates for data sparsity by extracting maximum value from available training examples. The modular architecture further provides significant data security benefits by restricting sensitive information processing to the isolated DL module, preventing confidential data from entering large-scale LLM training pipelines where privacy guarantees are difficult to enforce. The modular approach simultaneously reduces implementation costs, lowers technical barriers to specialized applications, and enhances data sovereignty critical advantage for domains with stringent regulatory compliance requirements or proprietary information concerns.

The DL-LLM Modular System fundamentally addresses three critical challenges facing current LLM implementations: training inefficiency, computational cost escalation, and dataset limitations. By optimizing the division of responsibilities between specialized AI components, the DL-LLM Modular System provides a scalable framework that enhances performance while simultaneously reducing resource requirements-establishing a new efficiency paradigm for advanced AI systems in practical deployment scenarios.

3.3 Future Plans

The ongoing research agenda will focus on three complementary paths to extend the capabilities of the DL-LLM Modular System while preserving its efficiency advantages.

- General Capability: Expand input diversity by incorporating audio samples with variable durations and acoustic characteristics, systematically evaluating how temporal variations affect classification performance. Additionally, expanding the modular system to incorporate Small Language Models (SLMs) will further support the models complete independence from server dependency and facilitate on-device deployability.

- Expanded Domain: Maintain the current models exceptional sample efficiency (i.e. ability to achieve high accuracy with minimal training examples) while expanding its classification domain.
- Prompting Engineering: Conduct comprehensive architecture optimization studies to identify the ideal hyperparameter configurations and component interactions that maximize end-to-end system performance. This systematic exploration will establish empirical guidelines for optimal DL-LLM integration across diverse application contexts.

4 Methods

4.1 Overview

The development of the DL-LLM Modular System followed a structured two-phase approach designed to create a seamless integration between specialized audio processing capabilities and LLM reasoning. 1) Audio DL model development for sound analysis, 2) Building the DL-LLM Modular System an integrated ecosystem accomplishing a unified communication system that combines the DL model and LLM.

4.2 Development Platforms & Tools

4.3 Building the Audio DL Model for Sound Analysis

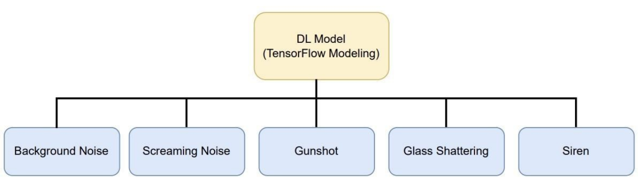


Fig. 5 Sound Classification Categories in the DL Model Using TensorFlow. This diagram illustrates the five distinct sound types that the DL model based on TensorFlow modeling is designed to recognize: Background Noise, Screaming Noise, Gunshot, Glass Shattering, and Siren. Audio samples were obtained from diverse sources such as YouTube.

For the sound classification component, Googles Teachable Machine was selected as the platform to develop the specialized deep learning model. This web-based tool offered significant advantages for this research, providing an accessible yet powerful environment to create a convolutional neural network (CNN) capable of distinguishing between five distinct acoustic events relevant to crime detection[30].

The development process began by establishing five distinct acoustic event classes as specified in Fig. 2: Background Noise, Screaming, Gunshot, Glass Shattering, and Siren. Multiple audio samples were recorded for each class in a controlled environment as illustrated in Table 12.

Table 11 Development Platforms and Tools. Details and documentation for each research tool, including specific LLM API versions utilized, are illustrated by the table.

Tool Name	Tool Description and Version
Google Colaboratory	Google Colab (Colaboratory) is a cloud-based Jupyter notebook environment that allows users to write and execute Python code in their browser[22]. This research used Google Colab to train the deep learning (DL) model through TensorFlow and build the DL-LLM Modular System.
Python	Python, a versatile programming language suitable for data science[23] was used for the training of the DL model based on TensorFlow modeling.
TensorFlow Lite (TFLite)	TFLite, a lightweight version of TensorFlow, provides a great method for running machine learning models on local laptops and training the DL model[24].
ChatGPT-4o	ChatGPT was the target LLM testing Public LLMs basic audio classification ability without its combination with the DL model .ChatGPT-4os 2025-03-27 version was utilized for this research[25].
Gemini 2.5 Pro	Google Gemini 2.5 Pro API (gemini-2.5-pro-preview-03-25:generateContent) was integrated into both the DL-LLM Modular System and baseline system to evaluate reasoning efficiency and improved domain-adaptation[26] This testing was repeated for the following LLMs illustrated in the following rows.
Claude 3.7 Sonnet	Claude 3.7 Sonnet API (claude-3-7-sonnet-20250219) was utilized for the research[27].
Deepseek R1	Deepseek R1 (deepseek-reasoner) API version 2025-01-19 was utilized for the research[28].
Grok 3	Grok 3 (grok-3) API April 2025 version was utilized for the research[29].

During real-time audio input, the Teachable Machine platform automatically converted the audio data into spectrogramstime-frequency representations that served as input to the neural network. Once validated, the model was exported in TensorFlow Lite format, capable of processing input audio as a float32[-1,44032] tensor (sampled at 44.1 kHz mono) and producing classification output as a float32[-1,5] tensor representing confidence scores for each of the five sound categories.

As shown in Fig.16, once the model is trained, it processes

Table 12 Quantity, Diversity, and Characteristics of Audio Samples Used for DL Model Training. Audio samples were balanced across categories, with the training dataset dynamically based the available audio sources and sample distributions for each sound class. Audio data was input by playing sounds directly from a speaker in a soundproof room, ensuring controlled recording conditions free from contamination.

Acoustic Sample	Number of Samples (1 sec per sample)	Sound Description
Background Noise	47	The background noise category consisted of quiet, ambient sounds to differentiate it from other sound classifications.
Screaming	56	The test utilized crowd screaming audio featuring persistent, intense shouting.
Gunshot	41	The gunshot audio featured a sharp, distinctive Tang firing sound typical of small arms discharge.
Glass Shattering	56	The glass shattering sound featured a breaking glass window, with the test focusing on the peak impact moment that produces the most characteristic shattering pattern.
Siren	34	The Siren sound had a repeating, oscillating pattern typical of police sirens and alternating frequency bands, suggesting the classic wee-woo pattern

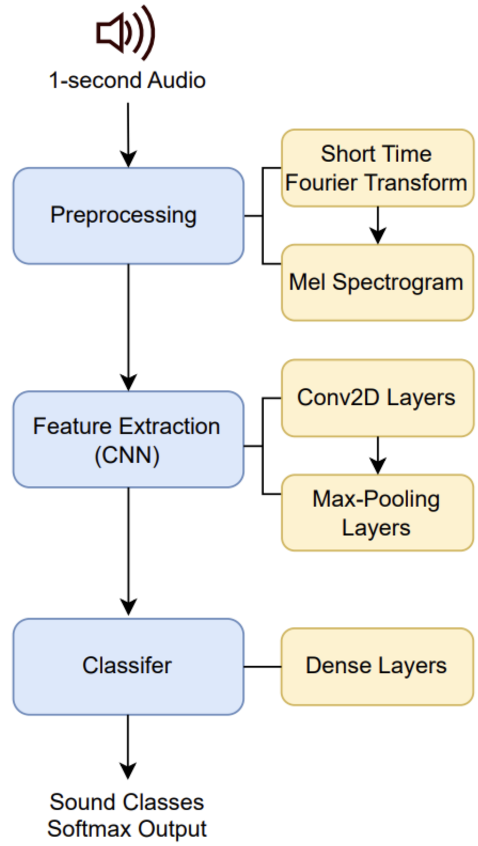


Fig. 6 Architecture of the Teachable Machine audio classification model. The model consists of three sequential stages: preprocessing (STFT and spectrogram generation), feature extraction (Conv2D and pooling layers), and classification (dense layers and softmax output). Each component is optimized to convert raw audio into meaningful representations for sound classification.

1-second audio clips by converting each into a two-dimensional spectrogram through a Short-Time Fourier Transform (STFT). STFT is a signal processing technique that decomposes an audio waveform into overlapping time-localized frequency components, enabling the representation of temporal and spectral features in a matrix format. This spectrogram - functionally equivalent to an image - is then fed into a compact convolutional neural network (CNN) for classification.

The CNN begins with alternating 2D convolutional (Conv2D) layers, which apply trainable filters to detect local patterns (e.g., harmonics, transients) in the spectrogram, and max-pooling layers, which downsample the feature maps to reduce spatial resolution while preserving dominant features. These convolutional blocks are followed by one or more fully connected (dense) layers, which integrate the spatially extracted features into higher-level representations. Dropout layers are interleaved with the dense layers to mitigate overfitting by randomly deactivating a subset of neurons during training.

The final output is produced by a softmax classification layer, which computes a normalized probability distribution over all predefined sound classes, indicating the models confidence for each class. The complete inference pipeline can be summarized as follows:

1-second audio $\rightarrow$ STFT spectrogram $\rightarrow$ Conv2D + max-pooling layers $\rightarrow$ dense + dropout layers $\rightarrow$ softmax output.

The trained Teachable Machine model was not further optimized or fine-tuned based on specific criteria. Instead, the TensorFlow Lite model downloaded directly from the Teachable Machine web interface was uploaded to Google Colab for use. The model utilized in this research can be viewed and downloaded via the following link: https://teachablemachine.withgoogle.com/models/CcUMQ3Lze/ The trained Teachable Machine model displayed a moderate accuracy as shown by Fig 5, the F1 Score results of the model.

Table 13 Performance Metrics of the DL Teachable Machine Model.

	Precision	Recall	F1-Score	Support
Glass Shattering	0.73	0.77	0.75	30
Gunshot	0.76	0.76	0.76	30
Screaming	0.68	0.59	0.63	30
Siren	0.63	0.61	0.62	30
Micro Avg	0.68	0.68	0.68	120
Macro Avg	0.7	0.68	0.69	120
Weighted Avg	0.7	0.68	0.69	120

Table 13 details the classification performance of the model on the four audio categories, evaluated on a test set where the support, or number of samples for each class, was 30. The results are presented in terms of precision, which measures the accuracy of the model's positive predictions, and recall, which measures the model's ability to identify all actual positive instances. The F1-score, the harmonic mean of precision and recall, is included to provide a single metric that balances the trade-off between false positives and false negatives.

The model demonstrates a moderate overall performance, achieving a macro average F1-score of 0.69. While this F1-score may not be optimal for real-world deployment, the performance was sufficient for this research, which used distinct audio samples to focus on the enhancement in an LLMs reasoning efficiency and domain adpatability when coupled with a specialized DL model.

Table 14 details the audio dataset used for evaluating the model. The evaluation set consists of 120 unique audio files, balanced equally with 30 samples for each of the four categories: Gunshot, Glass Shattering, Screaming, and Siren.

To ensure a robust test of generalization, each category was diversified with varied acoustic scenarios, such as including different firearm types for 'Gunshot' and multiple siren patterns for 'Siren'. Critically, these files were entirely separate from the training data to provide an unbiased measure of the model's performance on unseen material.

Table 14 Audio File Information Used for the training of the DL Teachable Machine Model.

Main Category	Audio File Description	Number of Samples
Gunshot	Standard small-arms Tang sound	15
	Rifle or shotgun discharge sounds	5
	Gunshots recorded outdoors with audible echo	5
	Distant or muffled gunshots	5
Glass Shattering	A breaking window pane	15
	A breaking glass cup	8
	A smashing bottle	7
Screaming	Crowd screaming	10
	Individual adult screams (mix of male/female)	10
	A child's scream	5
	Lower intensity or obscured screams	5
Siren	Classic "wee-woo" police sirens	15
	Ambulance or fire truck sirens (wail/yelp)	5
	Sirens with Doppler effect (passing vehicle)	5
	European-style high-low sirens	5

4.4 Building the Modular Architecture by Combining the DL Model with LLM

The trained audio classification model was integrated with different LLM models - Gemini 2.5 Pro, Claude 3.7 Sonnet, DeepSeek R1, and Grok-3 - to create the DL-LLM Modular System for crime sound detection. This integration formed the core of the system, enabling clear separation between the specialized audio processing and language reasoning components. The integration architecture consisted of two principal code modules, the Audio Preprocessing & Input Standardization module and the Inference and LLM Engine & Time Measurement module, as illustrated in the following section.

4.4.1 The Modular System Architecture

The first module of the two separate modules, (1) the Audio

Preprocessing Module, handles signal normalization and format standardization to meet the precise input specifications required by DL model, while the (2) the Time-Tracked DL Inference - LLM Response Module orchestrates the time-measured workflow between the DL classification system and subsequent LLM reasoning processes.

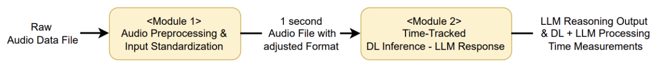


Fig. 7 Detailed view of the DL-LLM Modular System.

Module 1: Audio Preprocessing & Input Standardization The first module handles DL model initialization and audio data preparation, establishing the TensorFlow Lite interpreter for the custom sound classification model and configuring input tensors to match the required format (float32[-1,44032]).

Module 2: Time-Tracked DL Inference - LLM Response The second module manages the execution pipeline, including model inference, LLM integration, and performance (process time) measurement. When an audio file is processed, the TensorFlow Lite interpreter generates classification confidence scores across the five predefined sound categories. These structured outputs are then formatted into a prompt template and transmitted to LLMs. A high-precision timer is implemented within this module to measure the complete processing time, from initial model inference through API transmission to final response generation, providing quantifiable metrics for system evaluation.

4.4.2 Implementation of Module 1: Audio Preprocessing and Input Standardization

Module 1 handles the audio preprocessing for the DL-LLM Modular System. The code first uploads both the Teachable Machine DL model and target audio file, then standardizes the audio by resampling to 44.1 kHz using librosa. To meet Teachable Machines input requirements, the audio is precisely trimmed or zero-padded to 44032 samples (approximately one second). The preprocessing includes injecting minimal Gaussian noise (= 1e-5) to prevent mathematical errors during the DL models internal log-mel spectrogram computation. Finally, the audio is formatted as a float32 tensor with expanded dimensions and saved as 'input_tensor.npy' for Module 2's DL model inference.

4.4.3 Implementation of Module 2: Time-Tracked DL Inference - LLM Response

Module 2 implements the core DL-LLM integration workflow with precise performance measurement. The code begins by loading the preprocessed input tensor from Module 1 and automatically detecting the DL model file in the working directory. It initializes the TensorFlow Lite interpreter, allocates tensors, and verifies input/output dimensions for compatibility. During DL inference, the system uses nanosecond-precision timing (time.perf_counter_ns()) to capture exact processing duration. The raw confidence scores are

```python
# Module 1: Audio File Upload and Preprocessing

# Required libraries
!pip install librosa soundfile

from google.colab import files
import librosa
import numpy as np

# ---------- Upload Files ----------
print("Upload TFLite model:")
model_files = files.upload()  # Upload the TFLite model file (e.g., soundclassifier_with_metadata.tflite)

print("Upload audio file:")
audio_files = files.upload()  # Upload the audio file (e.g., unknown_audio_file.wav)

# ---------- Audio Preprocessing ----------
# Load the audio file (ensuring 44.1 kHz sample rate)
audio_path = list(audio_files.keys())[0]
y, sr = librosa.load(audio_path, sr=44100)

# If not done, Pad or trim to ~1 second (Teachable Machine expects ~44032 samples)
target_len = 44032
if len(y) < target_len:
    y = np.pad(y, (0, target_len - len(y)))
else:
    y = y[:target_len]

# Inject tiny Gaussian noise to avoid issues during log-mel processing
epsilon = 1e-5
y = y + np.random.normal(0, epsilon, size=y.shape)

# Create the input tensor for the TFLite model
input_tensor = np.expand_dims(y.astype(np.float32), axis=0)

# Save the preprocessed input tensor to file for Section 2
np.save('input_tensor.npy', input_tensor)

print("Preprocessing complete. Saved 'input_tensor.npy'.")
```

Fig. 8 Code Snippets for Module 1 of the DL-LLM Modular Architecture. This code snippet illustrates the initial workflow of the DL-LLM System, specifically the audio input standardization process. Although the audio file is pre-formatted to meet the required specifications (length, waveform, etc.), the standardization procedure is implemented as a safeguard against potential preprocessing errors. Complete implementation codes for each LLM are available in the shared Google Colab project (link provided at the end of the Methods section).

formatted into human-readable percentages for the five audio classes: Background Noise, Glass Shattering, Gunshot, Screaming, and Siren. The formatted results are then embedded into a structured prompt instructing the LLM to provide actionable responses. The system simultaneously measures LLM API response time using the same timing methodology. Finally, both DL and LLM processing times are computed in milliseconds, enabling quantitative analysis of the modular systems performance characteristics and demonstrating the efficiency gains from specialized task division.

4.4.4 Implementation of the Baseline System Using Gemini 2.5 Pro API: Time-Tracked LLM Inference for Audio Prediction Based on the Input

As illustrated in Figure 7, this code block invokes the LLM (Gemini 2.5 Pro) to process the identical audio file used by the DL-LLM Modular System, maintaining consistent audio content and formatting specifications (length, waveform, etc.). Here, the LLM is instructed to generate results in the same format as the Teachable Machine, listing potential labels with their respective probabilities. The code measures the LLM's processing time from the moment the API call is initiated with the prompt and audio file until the response is generated.

4.4.5 Implementation of the Baseline System Using Gemini 2.5 Pro API: Time-Tracked LLM Response for Sound Classification Based on the Audio Prediction

The LLM API sound prediction generated by the previous code snippet (Figure 9) is passed to a second LLM API, along with instructions to classify the sound based on the provided prediction metrics. The

Fig. 9 Code Snippets for Module 2 of the DL-LLM Modular Architecture. The code demonstrates the core workflow: specialized DL model performs audio classification, LLM provides interpretation and reasoning, with performance metrics quantifying system efficiency gains. Complete implementation codes for each LLM are available in the shared Google Colab project (link provided at the end of the Methods section).

time tracking logic measures processing duration from API call initiation until response generation. This LLM-LLM architecture replicates the DL-LLM Modular System workflow while relying solely on LLM capabilities rather than incorporating a specialized DL model for unfamiliar sound classification.

Below are the direct links to the modules designed using Google Colab:

- DL-LLMModularSystemUsingDeepseekR1: https://colab.research.google.com/drive/1Fik70Ne8wmYetTL0EiMuzv_XCKIDwP7Y?authuser=3#scrollTo=-5sqZCbsFOZu
- DL-LLMModularSystemUsingClaude3.7Sonnet: https://colab.research.google.com/drive/1pn3yPlrU1JEOfJDAROT-AvsH8z9nbWrF?authuser=3
- BaselineLLM-LLMModelUsingGemini2.5Pro: https://colab.research.google.com/drive/1QOgPhhifSClvhI33HSwH68b21hxpBXMn?authuser=3

Fig. 10 Gemini API Audio Analysis and Classification Code Snippets with Performance Tracking. The code demonstrates the initial workflow of the Baseline LLM-LLM System: the LLM (Gemini 2.5 Pro) simple audio prediction. Complete implementation is available in the shared Google Colab project (link provided at the end of the Methods section).

Fig. 11 Gemini API Audio Classification Results Analysis for Sound Prediction. The code demonstrates the continuing, or the final workflow of the Baseline LLM-LLM System: the LLM (Gemini 2.5 Pro) performs the audio classification. Complete implementation is available in the shared Google Colab project (link provided at the end of the Methods section).

- DL-LLMModelUsingGemini2.5Prohttps://colab.research.google.com/drive/1pvMwySxEZ8ZTbsGtnRTA1YMlb69X0Jpv?authuser=3
- DL-LLMModelUsingGrok-3https://colab.research.google.com/drive/1Lww88TbSfrwUeQfnWkBtamZ9QD8K7Jh2#scrollTo=

yzNfD6rftybt&uniqifier=2

5 Acknowledgements

I would like to express my sincere gratitude to Mr. Sim Jungwon for his invaluable guidance and mentorship throughout this research. His technical insights, methodological guidance, and consistent support were instrumental in overcoming key challenges and bringing this work to fruition.

I would also like to thank Mr. Evan Sayles for providing valuable feedback on my paper, particularly regarding statistical analysis and professional formatting standards that significantly enhanced the quality of this research.

References

1 P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal and A. Chadha, *arXiv preprint arXiv:2402.07927*, 2024.

2 T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever and D. Amodei, *arXiv preprint arXiv:2005.14165*, 2020.

3 C. Li and J. Flanigan, *arXiv preprint arXiv:2312.16337*, 2023.

4 Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang and H. Wang, *arXiv preprint arXiv:2312.10997*, 2023.

5 T. Dettmers, A. Pagnoni, A. Holtzman and L. Zettlemoyer, *arXiv preprint arXiv:2305.14314*, 2023.

6 K. Buchholz, *The extreme cost of training AI models*, 2024, https://www.forbes.com/sites/katharinabuchholz/2024/08/23/the-extreme-cost-of-training-ai-models/, Forbes.

7 TeamGPT, *How much did it cost to train GPT-4?*, 2024, https://team-gpt.com/blog/how-much-did-it-cost-to-train-gpt-4/.

8 M. Tatarek, *Costs and benefits of your own LLM*, 2023, https://medium.com/@maciej.tatarek93/costs-and-benefits-of-your-own-llm-79f58c0eb47f, Medium.

9 J. Cho, *AIs growing carbon footprint*, 2023, https://news.climate.columbia.edu/2023/06/09/ais-growing-carbon-footprint, Columbia Climate School News.

10 R. Robles, *Comparing fine-tuning optimization techniques: LoRA, QLoRA, DoRA, and QDoRA*, 2024, https://www.encora.com/insights/comparing-fine-tuning-optimization-techniques-lora-qlora-dora-and-qdora, Encora.

11 X. Xu, M. Li, C. Tao, T. Shen, R. Cheng, J. Li, C. Xu, D. Tao and T. Zhou, *arXiv preprint arXiv:2402.13116*, 2024.

12 R. Wang, Y. Gong, X. Liu, G. Zhao, Z. Yang, B. Guo, Z. Zha and P. Cheng, *arXiv preprint arXiv:2501.17116*, 2025.

13 S. Oliver and M. Lam, *Three proven strategies for optimizing AI costs*, 2024, https://cloud.google.com/transform/three-proven-strategies-for-optimizing-ai-costs, Google Cloud.

14 Y. Jin, J. Li, Y. Liu, T. Gu, K. Wu, Z. Jiang, M. He, B. Zhao, X. Tan, Z. Gan, Y. Wang, C. Wang and L. Ma, *arXiv preprint arXiv:2405.10739*, 2024.

15 G. Tyukin, *arXiv preprint arXiv:2404.05741*, 2024.

16 AWS, *Fine-tune large language models with reinforcement learning from human or AI feedback*, 2025, https://aws.amazon.com/blogs/machine-learning/fine-tune-large-language-models-with-reinforcement-learning-from-human-or-ai-feedback, AWS Machine Learning Blog.

17 H. Parthasarathy, *arXiv preprint arXiv:2408.13296*, 2024.

18 J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson *et al.*, *arXiv preprint arXiv:2204.14198*, 2022.

19 G. Tyukin, *arXiv preprint arXiv:2404.05741*, 2024.

20 Google, *Gemini API: Audio*, n.d., https://ai.google.dev/gemini-api/docs/audio.

21 Y. Yang, Y. Xu and L. Jiao, *arXiv preprint arXiv:2407.05347*, 2024.

22 Google, *Colab*, n.d., https://colab.google.

23 Python Software Foundation, *Python documentation: Blurb*, n.d., https://www.python.org/doc/essays/blurb/.

24 Google AI Edge, *TensorFlow Lite Runtime*, n.d., https://ai.google.dev/edge/litert.

25 OpenAI, *Model release notes*, 2024, https://help.openai.com/en/articles/9624314-model-release-notes.

26 Google AI for Developers, *Release notes: Gemini API changelog*, 2025, https://ai.google.dev/gemini-api/docs/changelog.

27 Anthropic, *Models Overview*, 2025, https://docs.anthropic.com/en/docs/about-claude/models/overview.

28 Deepseek, *Change Log*, 2025, https://api-docs.deepseek.com/updates.

29 X, *Release Notes*, 2025, https://docs.x.ai/docs/release-notes.

30 J. Chen, O. Firat, A. Bapna, M. X. Chen, W. Macherey and Z. Chen, *Teachable machine: Approachable web-based tool for exploring machine learning classification*, 2020, https://research.google/pubs/teachable-machine-approachable-web-based-tool-for-exploring-machine-learning-classification/, Google Research.