SONICRAG : HIGH FIDELITY SOUND EFFECTS SYNTHESIS BASED ON RETRIEVAL AUGMENTED GENERATION

Yu-Ren Guo, Wen-Kai Tai
Dept. of Computer Science and Information Engineering
National Taiwan University of Science and Technology
m11115115,wktai@mail.ntust.edu.tw

ABSTRACT

Large Language Models (LLMs) have demonstrated remarkable capabilities in natural language processing (NLP) and multimodal learning, with successful applications in text generation and speech synthesis, enabling a deeper understanding and generation of multimodal content. In the field of sound effects (SFX) generation, LLMs have been leveraged to orchestrate multiple models for audio synthesis. However, due to the scarcity of annotated datasets, and the complexity of temproal modeling. current SFX generation techniques still fall short in achieving high-fidelity audio. To address these limitations, this paper introduces a novel framework that integrates LLMs with existing sound effect databases, allowing for the retrieval, recombination, and synthesis of audio based on user requirements. By leveraging this approach, we enhance the diversity and quality of generated sound effects while eliminating the need for additional recording costs, offering a flexible and efficient solution for sound design and application.

Keywords Audio Synthesis · Large Language Model · Retrieval Augmented Generation

1 Introduction

Large language models (LLMs) have become an integral part of modern life, demonstrating exceptional performance in text-based tasks across various fields, such as natural language processing (NLP) [1, 2] and expert systems. In the field of speech synthesis, models like VITS[3] have achieved remarkable success. However, sound effects (SFX) synthesis remains a challenging problem due to:

- The lack of efficient and precise annotation methods for sound events.
- The complexity of modeling multiple overlapping sound events occurring simultaneously.

While LLMs can analyze textual descriptions to extract sound event sequences and break down tasks for sound synthesis [4, 5], the quality of audio generated by latent diffusion models (LDMs) such as AudioLDM still falls short of professionally recorded studio audio. Additionally, due to the nature of audio, humans are challenging to edit and reconstruct details as they wish in image editing.

To address these challenges, this paper proposes **SonicRAG**, integrates retrieval-augmented generation (RAG) [6] with LLMs' ability to analyze complex descriptions. By leveraging a small set of high-quality sound effects and describing sound tracks with abstract language—**Mixer Script**, our approach enables language models to synthesize diverse and user-intended sound effects, bridging the gap between AI-generated audio and professional-quality recordings.

2 Releated Work

2.1 Audio Generation

Depending on the characteristics of different sounds, the suitable sound synthesis methods may vary. For example, in the field of speech synthesis, VAE-based models such as VITS[3] have achieved excellent performance in text-to-

speech tasks. In music generation, Transformer-based models like AudioCraft[7] and MuseNet[8] have also produced promising results. Meanwhile, in sound effect generation, the current mainstream approach involves diffusion-based models, such as AudioLDM[9] and Stable Audio[10]. Although these methods are capable of generating content that aligns well with textual descriptions, they tend to struggle when dealing with more abstract or nuanced creative requirements.

2.2 Assistants Based on LLMs

In the era of AI-generated content (AIGC), many enterprises have demonstrated that large language models (LLMs) based on the Transformer architecture are capable of understanding and assisting in meeting human needs. Such as ChatGPT, DeepSeek [11, 12]help solve problems through conversational question-and-answer interactions. GitHub Copilot, powered by OpenAI Codex [13], analyzes engineers' code and requirements to derive solutions. Additionally, Microsoft Copilot [14]integrates LLMs with Microsoft Graph, further enhancing the user experience of office software. These applications prove that LLMs can serve as central decision-making hubs capable of analyzing multimodal inputs and generating instructions to accomplish a wide variety of tasks.

2.3 Prompt Engineering

Since LLMs typically contain billions of parameters, fine-tuning them for every specialized task would require enormous computational resources. Fortunately, the self-attention mechanism of LLMs allows them to infer user intent based on contextual information. This enables the use of prompt engineering techniques to achieve strong performance without the need for retraining. Brown *et al.* [15] proposed that when a model has a sufficiently large number of parameters, providing it with a small number of task examples can guide it toward the desired outcome. Additionally, Wei *et al.* [16] suggested that prompts can be designed to simulate human thinking processes, further enhance the model's performance on specialized tasks.

2.4 Retrieval-Augmented Generation

Although LLMs demonstrate the ability to retain knowledge, it remains infeasible to train an all-knowing model within a limited budget. Lewis *et al.* [6] shows that parametric and non-parametric memory doing postive interaction on knowledge-intensive tasks. This means that providing pre-query information along with the target prompts can improve the performance of LLMs. In our framework, LLMs analyze user's requirement, retrieval the audio assets, and generate audio sequence for synthesis sound effects.

3 Methodology

3.1 Mixer Script

Although LLMs can generate code for synthesizing sounds and execute the code to obtain audio, for sound designers and mixing engineers, understanding an unfamiliar programming language can be a significant hurdle. Moreover, when generating complex scripts, LLMs are prone to producing code that fails to execute reliably due to model limitations. To address this, we design an abstract scripting language—Mixer Script, which encapsulates audio signal processing methods. By leveraging method chaining to describe the parameters of each audio asset in a task, our approach reduces the difficulty of reading generated code and simplify the complexity of code generation for LLMs. The syntax of Mixer Script is defined in Figure 1 using EBNF (Extended Backus-Naur Form) [17], and the supported audio processing methods are defined in Table 1, As an example, the script for sound synthesis "coin dropped onto wooden table" generated by using **Mixer Script** and **Python**, is used to compare readability between Figure 2 and Figure 3.

```
                    EBNF of Mixer Script

<mixer_script>   ::= "<audio>" <statement>* "</audio>"
<statement>      ::= <string> <operation_chain> ";"
<operation_chain>::= <method_call>+
<method_call>    ::= "." <method_name> "(" <parameter_list> ")"
<parameter_list> ::= <parameter> ("," <parameter>)*
<parameter>      ::= <key> "=" <value> | <value>
```

Figure 1: *Extended Backus-Naur Form of Mixer Script*

Mehtods	Parameters	Adjusts
Volume	targetLUFS	volume to target loudness
Compressor	threshold, ratio, attack_ms, release_ms	dynamic of sound
Reverb	room_size, dry_wet	adding the Reverb Effect
PeakFilter	frequency, q_facter, gain	voulme of specified frequency
LowPassFilter	frequency	cutoff volume over target frequency
HighPassFilter	frequency	cutoff volume below target frequency
StartAt	at	start time of sound event
StopAt	at, fade_out_duration	fade out the sound event

Table 1: Audio Processing Methods supported by MixerScript

Mixer Script: coin dropped to wooden table

```
<audio>
 "coin collide wood".Volume(-14).StartAt(0.3);
 "coin collide wood".Volume(-17)
                    .PeakFilter(frequency=2000, q_factor=0.7, gain=3)
                    .StartAt(0.5);
 "coin whirl on wood".Volume(-14)
                    .Compressor(threshold=-21, ratio=2.4,
                              attack_ms=20, release_ms=500)
                    .StartAt(0.7);
</audio>
```

Figure 2: *The sound of "coin dropped to wooden table" described by Mixer Script.*

Python:coin dropped to wooden table

```
from AudioProcess import LoadAudio,Silence,PeakFilter,Compressor,Mix

#loading audio from dir
Audio1 = loadAudio("/filepath/coin collide wood.mp3")
Audio2 = loadAudio("/filepath/coin collide wood.mp3")
Audio3 = loadAudio("/filepath/coin whirl on wood.mp3")

#audio editing
Audio1 = Audio1**10*(-14/20)
Audio1 = Silence(Audio1.sr*0.3) + Audio1
Audio2 = Audio2**10*(-17/20)
Audio2 = PeakFilter(Audio2,2000,0.7,3)
Audio2 = Silence(Audio2.sr*0.5) + Audio2
Audio3 = Audio3**10*(-14/20)
Audio3 = Compressor(Audio3,-21,2.4,20,500)
Audio3 = Silence(Audio3.sr*0.7) + Audio3

#mix it
processed_audio = Mix([Audio1,Audio2,Audio3])
```

Figure 3: *The python code of "coin dropped to wooden table", using pre define library "AudioProcess" for signal processing.*

3.2 Metadata of Sound Events

While LLMs pretrained with multimodal have the ability to process audio tokens, but most that are optimized for speech and exhibit limitations in understanding the characteristics of sound events. For example, they may struggle to interpret pitch or complex acoustic scenes. Therefore, it becomes necessary to use textual representations to help LLMs comprehend the metadata of sound events. In our framework, we identify four key parameters essential to sound design, namely **Loudness**, **Voice onset time**, **Pitch**, and **Duration**. By packaging this information together with a textual

description into a unified sound object, we enable large language models to infer the characteristics of corresponding sound. This design also allows our framework to leverage LLMs trained solely on textual data for sound generation tasks. Figure 4 is an example of sound object.

```
┌─────────────────────────────────────┐
│        Unified Sound Object          │
└─────────────────────────────────────┘
┌─────────────────────────────────────┐
│ "SoundEvents":                       │
│ [                                    │
│     {                                │
│         "Name":"coin collide wood",  │
│         "Metadata":{                 │
│             "LUFS":-23,              │
│             "VOT":0.2,               │
│             "Pitch":-,               │
│             "Duration":0.4           │
│         }                            │
│     },                               │
│     ...                              │
│ ]                                    │
└─────────────────────────────────────┘
```

Figure 4: *Example of unified sound object. In the sound event list, each sound would be packaged to a object with it's name and metadata.*

3.3 Context-Aware Retrieval

In Retrieval-Augmented Generation (RAG), prompts serve not only to drive LLMs but also to retrieve essential reference data. However, in the task of sound sequence synthesis, each user input does not necessarily require retrieving for new sound assets, such as re-synthesis or fine-tuning may not demand additional queries. While we could ignore this and perform retrieval regardless, excessive and irrelevant context could negatively impact the performance of LLMs.

To overcome this, we have implemented a mechanism that allows LLMs to autonomously analyze the dialogue context and determine whether new sound assets are needed. This approach offers two key advantages: (1) Sparing complex rule-based scripts to analyze each user input. (2) It allows query terms to be formulated in a specified language, preventing cross-language queries from reducing the accuracy of the vector database.

3.4 Framework

In our proposed SonicRAG framework, the generation of an audio synthesis script $\mathcal{S}$ is modeled as a Markov decision process [18]:

$$\text{SonicRAG} = P(\mathcal{S}_t \mid \mathcal{S}_{t-1}, \mathcal{P}, \mathcal{M}, \mathcal{L}, \mathcal{R}), \tag{1}$$

Where $\mathcal{P}$ is the user prompt describing the desired sound, $\mathcal{M}$ represents predefined audio processing methods, $\mathcal{L}$ is language model which is used to drive the system, and $\mathcal{R}$ consists of retrieved audio assets, which is modeled as:

$$\text{Retrieval} = P(\mathcal{R}_t \mid \mathcal{R}_{t-1}, \mathcal{P}, \mathcal{L}), \tag{2}$$

It means that whether retrieval is updated depends on current state and the LLM's analysis of prompt. The script generation process is context-aware, incorporating both retrieval-augmented information and domain-specific synthesis methods to iterate listening experience and relevance of the generated sounds.

4 Experiments

4.1 Setup

For our experiments, we selected gpt-4o-audio-preview as the engine to drive SonicRAG. This large language model supports audio as input modality, meaning it can process audio tokens directly. Without relying on speech recognition, users can describe the desired sound effects using spoken descriptions, including onomatopoeia. We recorded a series of coin-related sound with sampling format of 48kHz/24bit for assets and named the files according to their corresponding events, the filenames are also stored in a vector database to facilitate retrieval based on specific queries.

4.2 Case study: Sound Design Workflow

Sound design plays a crucial role in multimedia industries such as film and game design. Well-crafted sound effects enhance immersion, elevating the overall experience. For sound designers who have already built extensive sound libraries, the key challenge is efficiently analyzing creative intent and transforming imagination into high-quality sound effects.

SonicRAG addresses this need by enabling interactive dialogue, allowing for iterative refinement of the designer's vision and seamless integration of existing assets into a cohesive output. The following demonstrates SonicRAG in real-world applications:
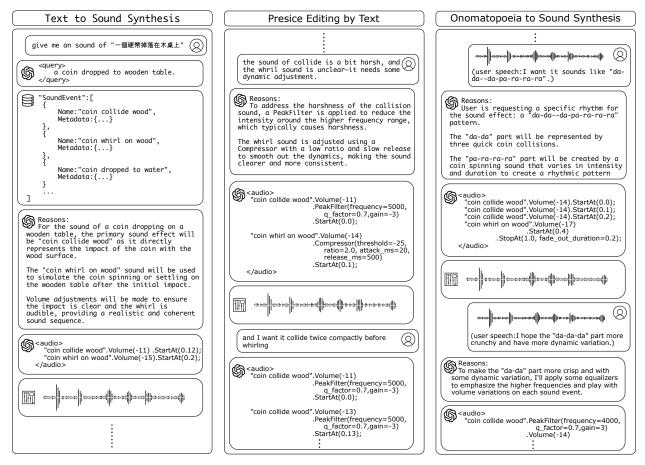
Text to Sound Synthesis

give me an sound of "一個硬幣掉落在木桌上"

```
<query>
    a coin dropped to wooden table.
</query>
```

```
"SoundEvent":[
    {
        Name:"coin collide wood",
        Metadata:{...}
    },
    {
        Name:"coin whirl on wood",
        Metadata:{...}
    },
    {
        Name:"coin dropped to water",
        Metadata:{...}
    }
    ...
]
```

Reasons:
For the sound of a coin dropping on a wooden table, the primary sound effect will be "coin collide wood" as it directly represents the impact of the coin with the wood surface.

The "coin whirl on wood" sound will be used to simulate the coin spinning or settling on the wooden table after the initial impact.

Volume adjustments will be made to ensure the impact is clear and the whirl is audible, providing a realistic and coherent sound sequence.

```
<audio>
    "coin collide wood".Volume(-11) .StartAt(0.12);
    "coin whirl on wood".Volume(-15).StartAt(0.2);
</audio>
```

Presice Editing by Text

the sound of collide is a bit harsh, and the whril sound is unclear—it needs some dynamic adjustment.

Reasons:
To address the harshness of the collision sound, a PeakFilter is applied to reduce the intensity around the higher frequency range, which typically causes harshness.

The whirl sound is adjusted using a Compressor with a low ratio and slow release to smooth out the dynamics, making the sound clearer and more consistent.

```
<audio>
    "coin collide wood".Volume(-11)
                    .PeakFilter(frequency=5000,
                        q_factor=0.7,gain=-3)
                    .StartAt(0.0);

    "coin whirl on wood".Volume(-14)
                    .Compressor(threshold=-25,
                        ratio=2.0, attack_ms=20,
                        release_ms=500)
                    .StartAt(0.1);
</audio>
```

and I want it collide twice compactly before whirling

```
<audio>
    "coin collide wood".Volume(-11)
                    .PeakFilter(frequency=5000,
                        q_factor=0.7,gain=-3)
                    .StartAt(0.0);

    "coin collide wood".Volume(-13)
                    .PeakFilter(frequency=5000,
                        q_factor=0.7,gain=-3)
                    .StartAt(0.13);
```

Onomatopoeia to Sound Synthesis

(user speech:I want it sounds like "da-da--da-pa-ra-ra-ra".)

Reasons:
User is requesting a specific rhythm for the sound effect: a "da-da--da-pa-ra-ra-ra" pattern.

The "da-da" part will be represented by three quick coin collisions.

The "pa-ra-ra-ra" part will be created by a coin spinning sound that varies in intensity and duration to create a rhythmic pattern

```
<audio>
    "coin collide wood".Volume(-14).StartAt(0.0);
    "coin collide wood".Volume(-14).StartAt(0.1);
    "coin collide wood".Volume(-14).StartAt(0.2);
    "coin whirl on wood".Volume(-17)
                    .StartAt(0.4)
                    .StopAt(1.0, fade_out_duration=0.2);
</audio>
```

(user speech:I hope the "da-da-da" part more crunchy and have more dynamic variation.)

Reasons:
To make the "da-da" part more crisp and with some dynamic variation, I'll apply some equalizers to emphasize the higher frequencies and play with volume variations on each sound event.

```
<audio>
    "coin collide wood".PeakFilter(frequency=4000,
                        q_factor=0.7,gain=3)
                    .Volume(-14)
```

Figure 5: *Behavior of SonicRAG in diffrent working situation. Left side shows how is RAG and Mixer enable LLM to synthis audio. Central depicts adjustability of framework. Right part exhibits multimodel LLM can understand onomatopoeias and represent it to other media.*

4.3 Comparision

SonicRAG can reuse pre-recorded sound assets and synthesize sound effects through Mixer Script, giving it an advantage in both sound quality and precise sound adjustment. Additionally, by leveraging multimodal training, LLMs can be further extended to achieve speech-to-sound and even onomatopoeia-to-sound generation.

Due to the limitations of neural network models, previous approaches had to downsample the signal to around 16kHz or use spectrogram-based training and generation.While this yields satisfactory results for speech synthesis, the loss of details caused by aliasing becomes more apparent in sound effects. Precise editing means user could adjust components detail (e.g. Frequency, Dynamics and Voice onset time) of sound. Audio generation based on Latent Diffusion Models remains difficult to control. Although LLMs can interpret onomatopoeias and abstract concepts, they still struggle to consistently produce the desired results when applied to systems like WavCraft and WavJourney. Table 2 compares SonicRAG with existing approaches.

Method	Native sampling	Precise editing by text	onomatopoeia to sound
AudioGen [7]	✗	✗	✗
AudioLDM [9]	✗	✗	✗
WavCraft [19]	✗	✗/✓	✗/✓
WavJourney [20]	✗	✗/✓	✗/✓
SonicRAG$_{(ours)}$	✓	✓	✓

Table 2: *Comparison of capabilities between SonicRAG and recent audio generation methods.*

4.4 Specialized Generating

In this section, we present a collection of synthesis prompts related to coin sounds, such as "Coin dropped onto a wooden table" and "Multiple coins jingling together in a person's hand". We compare the results generated by our method with those from previous approaches using several evaluation metrics: Fréchet Audio Distance (FAD) [21], It measures the similarity between generated and real audio distributions based on deep feature embeddings, and we use ARCA23K[22] for the ground truth set. Contrastive Language-Audio Pretraining Score (CLAP)[23, 24], evulates semantic correlation between natural laguage and audio. Signal-to-Noise Ratio (SNR), a traditional audio quality metric that quantifies the amount of desired signal relative to background noise. We report the average comparison results for each method based on this metric.

Methods	FAD ↓	CLAP ↑	SNR ↑
AudioGen [7]	32.1	0.48	29.1
AudioLDM [9]	22.3	0.44	40.4
StableAudio [10]	**21.4**	0.56	37.0
WavCraft [19]	25.8	0.32	53.8
WavJourney [20]	27.9	0.44	40.8
SonicRAG$_{(ours)}$	25.5	**0.67**	**88.2**

Table 3: *Evaluation results on Specialized Generating*

5 Conclusion

5.1 Summary

In this paper, we proposed SonicRAG, a retrieval-augmented generation (RAG) framework tailored for sound synthesis. Unlike conventional methods that rely solely on pre-trained generative models or manual sound retrieval, SonicRAG enables an interactive workflow where users provide natural language prompts, and the system dynamically retrieves and mixes audio assets to generate high-quality sound effects.

Our experimental results demonstrate that SonicRAG achieves superior performance in requirement accuracy and synthesis flexibility compared to existing methods. By leveraging Mixer Script for structured audio translation, we provide sound designers with greater control over the final output while maintaining ease of use. Furthermore, our approach reduces the reliance on extensive retraining, making it a scalable solution for real-world applications.

Overall, SonicRAG presents a novel and practical approach to AI-assisted sound design, bridging the gap between retrieval-based and generative methodologies while offering an intuitive interaction model for creative professionals.

5.2 Limitation

SonicRAG provide user an creative and interactive interface for synthesis high fidelity sound effect. But it still have limitations: 1): Infrastructure: the framework needs user already have text-relational audio files for assets. 2): Perception: Due to different sense of hearing, capability of effects provided by Mixer Script may not enough for everyone. 3): Efficiency: it require larger size LLMs such as GPT-4o for comprehend abstract tasks.

5.3 Future Work

Use different sounds to ensemble euphonious results is a complex problem that encompasses multiple disciplines like SEC (sound event classfication), SED (sound event detection with timestamp) and ASC (Acoustic sence classification). It isn't solved effectively in textual-based methods so far. Future research could explore using task-specific pre-trained embeddings to assist in fine-tuning LLMs, optimizing their performance for sound synthesis.

References

[1] Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67, 2019.

[2] Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Transactions on Knowledge Discovery from Data*, 18:1 – 32, 2023.

[3] Jaehyeon Kim, Jungil Kong, and Juhee Son. Conditional variational autoencoder with adversarial learning for end-to-end text-to-speech. *ArXiv*, abs/2106.06103, 2021.

[4] Rongjie Huang, Mingze Li, Dongchao Yang, Jiatong Shi, Xuankai Chang, Zhenhui Ye, Yuning Wu, Zhiqing Hong, Jia-Bin Huang, Jinglin Liu, Yixiang Ren, Zhou Zhao, and Shinji Watanabe. Audiogpt: Understanding and generating speech, music, sound, and talking head. *ArXiv*, abs/2304.12995, 2023.

[5] Zixuan Wang, Yu-Wing Tai, and Chi-Keung Tang. Audio-agent: Leveraging llms for audio generation, editing and composition. *ArXiv*, abs/2410.03335, 2024.

[6] Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. *ArXiv*, abs/2005.11401, 2020.

[7] Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, and Alexandre Défossez. Simple and controllable music generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[8] OpenAI. Musenet, April 2019. Accessed: 2025-04-16.

[9] Haohe Liu, Zehua Chen, Yiitan Yuan, Xinhao Mei, Xubo Liu, Danilo P. Mandic, Wenwu Wang, and Mark D. Plumbley. Audioldm: Text-to-audio generation with latent diffusion models. In *International Conference on Machine Learning*, 2023.

[10] Zach Evans, Julian Parker, CJ Carr, Zack Zukowski, Josiah Taylor, and Jordi Pons. Stable audio open. *ArXiv*, abs/2407.14358, 2024.

[11] OpenAI Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, et al. Gpt-4 technical report. 2023.

[12] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Jun-Mei Song, Ruoyu Zhang, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *ArXiv*, abs/2501.12948, 2025.

[13] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *ArXiv*, abs/2107.03374, 2021.

[14] Microsoft. Introducing Microsoft 365 Copilot — your copilot for work, 2023. Accessed: 2024-04-28.

[15] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Ma teusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.

[16] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, F. Xia, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903, 2022.

[17] ISO 14977:1996: Information technology – Syntactic metalanguage – Extended BNF. International Organization for Standardization, 1996. https://www.iso.org/standard/26153.html.

[18] Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[19] Jinhua Liang, Huan Zhang, Haohe Liu, Yinzhi Cao, Qiuqiang Kong, Xubo Liu, Wenwu Wang, Mark D. Plumbley, Huy Phan, and Emmanouil Benetos. Wavcraft: Audio editing and generation with large language models. 2024.

[20] Xubo Liu, Zhongkai Zhu, Haohe Liu, Yiitan Yuan, Meng Cui, Qiushi Huang, Jinhua Liang, Yin Cao, Qiuqiang Kong, Mark D. Plumbley, and Wenwu Wang. Wavjourney: Compositional audio creation with large language models. *ArXiv*, abs/2307.14335, 2023.

[21] Kevin Kilgour, Mauricio Zuluaga, Dominik Roblek, and Matthew Sharifi. Fréchet audio distance: A metric for evaluating music enhancement algorithms. *ArXiv*, abs/1812.08466, 2018.

[22] T. Iqbal, Y. Cao, A. Bailey, M. D. Plumbley, and W. Wang. ARCA23K: An audio dataset for investigating open-set label noise. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2021 Workshop (DCASE2021)*, pages 201–205, Barcelona, Spain, 2021.

[23] Benjamin Elizalde, Soham Deshmukh, Mahmoud Al Ismail, and Huaming Wang. Clap: Learning audio concepts from natural language supervision, 2022.

[24] Feiyang Xiao, Jian Guan, Qiaoxi Zhu, Xubo Liu, Wenbo Wang, Shuhan Qi, Kejia Zhang, Jianyuan Sun, and Wenwu Wang. A reference-free metric for language-queried audio source separation using contrastive language-audio pretraining. In *Proceedings of Detection and Classification of Acoustic Scenes and Events (DCASE) Workshop*, 2024.