SonoCraftAR: Towards Supporting Personalized Authoring of Sound-Reactive AR Interfaces by Deaf and Hard of Hearing Users

Jaewook Lee
University of Washington
Davin Win Kyi
University of Washington
Leejun Kim
University of Washington
Jenny Peng
University of Washington
Gagyeom Lim
Dong-A University
Jeremy Zhengqi Huang
University of Michigan
Dhruv Jain
University of Michigan
Jon E. Froehlich
University of Washington

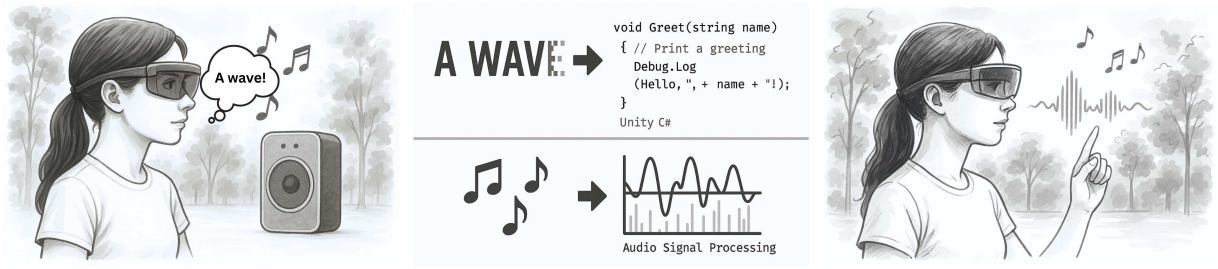

Figure 1: *SonoCraftAR* empowers Deaf and hard-of-hearing (DHH) users to author personalized, sound-reactive interfaces in wearable augmented reality (AR). Users type a desired UI description, which a multi-agent LLM pipeline converts into procedurally generated, animated visualizations. Our prototype system performs real-time audio signal processing to extract features (*i.e.,* dominant frequency), which drive the animations and make the visualizations respond dynamically to sound.

Abstract

Augmented reality (AR) has shown promise for supporting Deaf and hard-of-hearing (DHH) individuals by captioning speech and visualizing environmental sounds, yet existing systems do not allow users to create personalized sound visualizations. We present *SonoCraftAR*, a proof-of-concept prototype that empowers DHH users to author custom sound-reactive AR interfaces using typed natural language input. SonoCraftAR integrates real-time audio signal processing with a multi-agent LLM pipeline that procedurally generates animated 2D interfaces via a vector graphics library. The system extracts the dominant frequency of incoming audio and maps it to visual properties such as size and color, making the visualizations respond dynamically to sound. This early exploration demonstrates the feasibility of open-ended sound-reactive AR interface authoring and discusses future opportunities for personalized, AI-assisted tools to improve sound accessibility.

Index Terms: Augmented reality, AI, large language model, authoring, sound awareness, deaf and hard of hearing

1 Introduction

Augmented reality (AR) has shown promise as an assistive technology for people who are Deaf and hard of hearing (DHH), including for real-time captioning [8, 11, 12, 14, 19, 20, 22] and visualizing non-speech sounds [8, 11, 13]. While research has explored how to personalize sound recognition models using advances in deep learning [2, 6, 7, 15], limited prior work exists on personalizing assistive sound visualizations. One recent system, *SoundWeaver*, automatically adjusts AR interfaces based on a user's surroundings and intent [11]; however, it does not let users actively author new visualizations (*e.g.,* by typing "*a wave*") and instead focuses on automatic UI adaptation. While these systems offer valuable support, they remain largely preconfigured and do not yet allow DHH users to create their own personalized, sound-reactive AR interfaces.

Empowering disabled individuals to design their own tools is a crucial step toward inclusive technology [9, 24]. Recent advances in generative AI open new possibilities for supporting user-driven AR creation. For example, several systems enable XR scene composition through natural language using large language models (LLMs) [3, 4, 16, 26, 28]. Some also support simple movements [10, 17, 18, 27], though generating animations—especially those that respond to real-world sensory input—remains challenging. Critically, no prior XR authoring tool has explored open-ended AR interface creation for making sound more accessible.

We introduce *SonoCraftAR*, a novel proof-of-concept prototype that empowers DHH users to author personalized, sound-reactive AR interfaces through typed natural language input. Built for the *Microsoft HoloLens 2*[1], SonoCraftAR combines real-time audio signal processing with LLM agents to generate responsive 2D AR UIs that visualize ambient sound. To support flexible authoring, our system includes three *o3* [21] agents: a *Prompt Enhancement* agent that supplements user input with UI accessibility and design guidelines; a *Code Generation* agent that outputs a *Unity*[2] C# script using code documentation and examples; and a *Code Checker* agent that detects and resolves compile-time errors. UIs are rendered using *Roslyn*[3] for runtime code compilation and built with the *Shapes*[4] vector graphics library, which provides fine-grained control over visual properties for animations. To equip the LLM with relevant prior knowledge, we crawled the Shapes documentation and in-

[1] https://learn.microsoft.com/en-us/hololens/hololens2-hardware
[2] https://unity.com
[3] https://assetstore.unity.com/packages/tools/integration/roslyn-c-runtime-compiler-142753
[4] https://assetstore.unity.com/packages/tools/particles-effects/shapes-173167

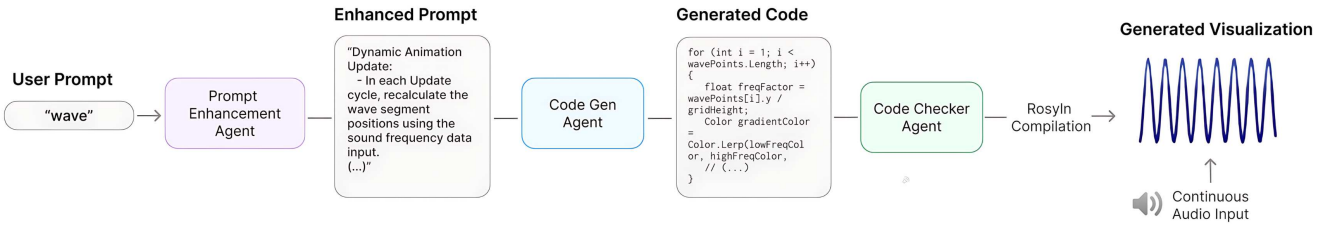

Figure 2: System overview of SonoCraftAR. A typed user prompt is first expanded by the *Prompt Enhancement* agent into structured implementation guidelines. The *Code Generation* agent then produces a Unity C# script that uses the Shapes vector graphics library, which is checked for compilation errors by the *Code Checker* agent. The finalized code script is compiled at runtime with Roslyn and rendered in AR. A real-time audio processing server continuously computes the dominant frequency of incoming sound, which drives the visualization's animations.

cluded it as part of the LLM system prompt. Interfaces respond to sound features such as dominant frequency by adjusting properties like shape size, color, and saturation. Together, these components enable open-ended authoring of sound-reactive AR interfaces tailored to DHH users' preferences and needs.

In this workshop paper, we describe the SonoCraftAR system and discuss its limitations and future opportunities for enabling hyper-personalized, sound-reactive UI authoring for DHH users.

2 The SonoCraftAR System

We begin by describing how we enabled LLMs to utilize a vector graphics library, followed by how LLMs are prompted to generate code that defines the structure and behavior of AR sound interfaces, which are then made sound-reactive through audio signal processing. By enabling Deaf and hard of hearing (DHH) users to author their own sound visualizations, *SonoCraftAR* facilitates more personalized representations—allowing users to perceive and interpret sound in ways that reflect their individual preferences.

2.1 Enabling LLM-Based Procedural UI Rendering

To support open-ended, sound-reactive AR interface generation, we required a rendering system that not only produces visually compelling UIs but also allows fine-grained control over individual components for animation. While recent work has explored generating 2D assets using diffusion-based models [23, 25], such outputs are poorly suited for responding to real-time sensory input like sound. In contrast, vector graphics enable procedural rendering, allowing properties such as color, size, and position to be modified dynamically at runtime based on audio features.

We selected *Shapes*[4], a real-time vector graphics library for *Unity*[2], to support code-driven UI rendering. Shapes enables procedural drawing of primitives such as lines, polygons, and arcs, which can be composed into more complex 2D elements. This allowed us to treat each UI as a collection of smaller, independently controllable components that can be updated in real time based on audio input—producing animated visualizations. For example, our system can pulse element size in sync with dominant frequency.

However, writing correct Shapes-based code requires familiarity with its custom C# API, which differs from Unity's default rendering primitives. To make this interface usable by LLMs, we crawled the official Shapes documentation[5] using *BeautifulSoup*[6] to extract its HTML source, then converted it to markdown using *html-to-markdown*[7]. The resulting document was reviewed by two researchers and included in the LLM system prompt as a reference guide. This approach enabled LLMs to generate code that accurately uses the Shapes API.

[5] https://acegikmo.com/shapes/docs
[6] https://www.crummy.com/software/BeautifulSoup/
[7] https://html-to-markdown.com

2.2 System Implementation

SonoCraftAR is implemented on the *Microsoft HoloLens 2*[1] using Unity 2022.3.52f1 and the *Mixed Reality Toolkit (MRTK)* 2.8.3[8]. The system employs a multi-agent LLM pipeline to expand user prompts, generate sound-reactive UI code, and check for compile-time errors. All prompts used are included in the Appendix. The resulting interface is rendered at runtime and responds dynamically to the dominant frequency of incoming sound via continuous audio signal processing. Because the HoloLens does not support runtime script compilation, SonoCraftAR includes a Windows laptop running the Unity Editor, which compiles and executes the generated scripts. We then use Holographic Remoting[9] to stream the resulting holographic content to the HoloLens 2 in real time for viewing. We detail system components below.

Understanding User Input. Modern speech transcription tools are typically trained on speech data from non-DHH speakers and often struggle to accurately transcribe speech from DHH users [5]. To address this limitation, SonoCraftAR supports typed natural language input, allowing users to provide prompts with reduced risk of recognition errors.

Expanding User Prompt. Users provide typed descriptions of desired sound visualizations. This input is passed to the *Prompt Enhancement* agent, which uses OpenAI's *o3* [21] to convert it into structured implementation instructions. The enhanced prompt adds details on how the visualization should look and animate (*e.g.,* pulsing size, shifting color, varying saturation or thickness) and specifies how to assemble relevant Unity and Shapes API methods. This enriched prompt lets the next agent focus on generating accurate code rather than simultaneously making design decisions—a separation that empirically reduced hallucinations.

Script Generation. The enhanced prompt is passed to the *Code Generation* agent, which uses one-shot prompting with the Shapes library documentation, a short Unity C# checklist (*e.g.,* namespaces), a template, and an example output to produce an appropriate script. Before doing so, it determines whether the user's query is for a new output or an iteration. The agent receives the previous prompt and code but is instructed to use this context only for follow-up queries, where it edits the existing visualization; otherwise, it ignores this context and creates a new script. The generated code also includes a function callable by the *Roslyn*[3] runtime compiler that takes real-time sound data as input, allowing the UI to dynamically respond to audio features.

Checking for Errors. After a script is generated, it is compiled, and any resulting errors are passed to the *Code Checker* agent along

[8] https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2
[9] https://learn.microsoft.com/en-us/windows/mixed-reality/develop/native/holographic-remoting-overview

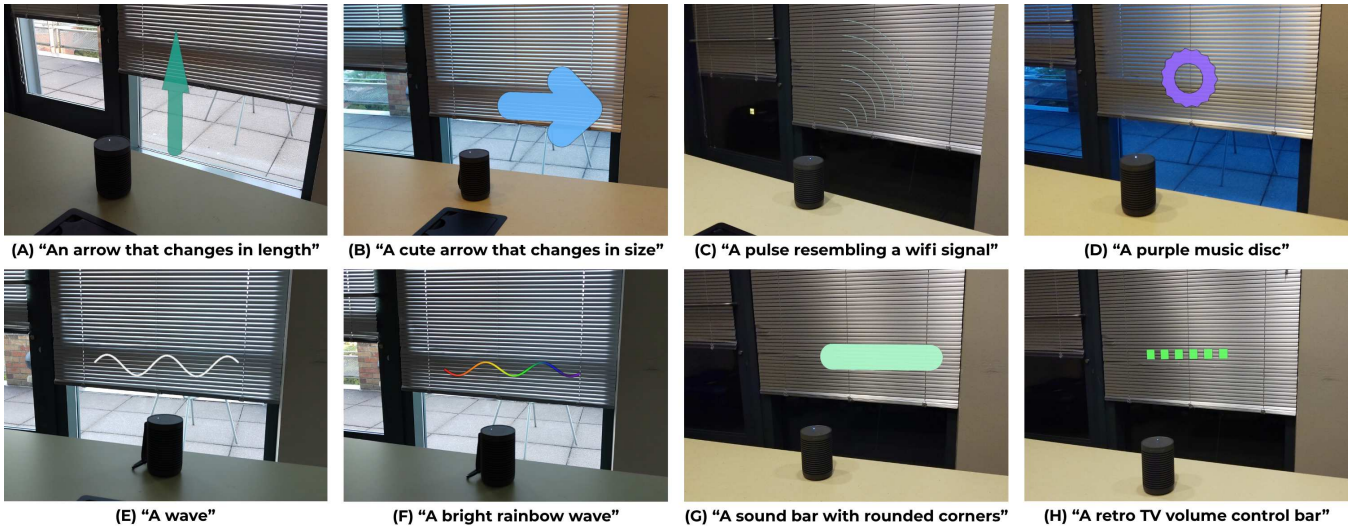

Figure 3: Eight example sound-reactive interfaces created with SonoCraftAR. The designs include arrows, waves, pulsing arcs, sound bars, and more. Examples A–C were inspired by GlassEar [13], while D–H showcase ideas proposed by the research team. These visualizations react to the dominant frequency of music playing through a speaker.

with the code and user prompt. Leveraging the Shapes documentation and a list of common Unity C# errors (*e.g.,* floats must end with ‘f’), the agent fixes compile-time issues and verifies that the code matches the user’s request. This final step improves reliability by catching errors missed during generation and producing a corrected script ready for deployment.

Rendering in AR. Finally, the generated script is compiled at runtime with Roslyn and rendered in AR through Holographic Remoting. Roslyn continuously supplies real-time sound data (*i.e.,* the dominant frequency value) to the script, enabling the visualization to react dynamically to incoming audio.

Audio Signal Processing. To extract sound features in real time, SonoCraftAR runs a Python server that samples stereo audio on a Windows laptop at 48 kHz and processes it in 100 ms non-overlapping chunks. The signal is first converted to mono and passed through a Hanning window to reduce spectral leakage. A real-valued Fast Fourier Transform (FFT) implemented with NumPy[10] is then applied to compute frequency-domain magnitudes, from which the dominant frequency—the peak-magnitude component within 20-8000 Hz (covering most speech and environmental sounds while excluding low-frequency rumble and high-frequency noise)—is extracted. This value is normalized to a 0–10 scale using logarithmic mapping, enabling smoother visual scaling in UI animations. The result is sent to the Unity Editor via a WebSocket connection. To keep the proof-of-concept system simple while demonstrating sound-reactivity, we chose to represent only the dominant frequency, which is quick to compute and often matches the perceived pitch of tonal sounds like music.

Latency. To evaluate runtime performance, we ran the system 50 times using the prompt “*a sound wave*” and observed an average generation time of 36.6 ± 13.9 seconds to produce a sound-reactive visualization.

Example Interaction. Suppose a user types “*a wave*”. SonoCraftAR could generate a white waveform visualization (Figure 3E) that reacts to music playing from a speaker. They can then follow up with “*make it rainbow-colored*”, to which our system could update the wave with a rainbow gradient (Figure 3F).

[10] https://numpy.org/doc/stable/reference/generated/numpy.fft.rfft.html

2.3 Example Creations

We demonstrate several sound-reactive AR interfaces authored with SonoCraftAR, including an arrow that changes in length or width, pulsing arcs, a rainbow-colored wave, and a retro-style TV volume bar. Some of these recreate UIs proposed in *GlassEar* [13], which outlined a design space for AR sound visualizations. Each example reacts dynamically to Beyoncé’s *Love on Top*, chosen for its varying dominant frequency, playing from a nearby speaker. Figure 3 shows all example creations.

3 Discussion and Future Work

SonoCraftAR is an early proof-of-concept that explores how DHH users can author personalized, sound-reactive AR interfaces. In this section, we outline the system’s limitations and highlight opportunities for future work.

3.1 Personalization vs. Effective Sound Representation

There is a potential trade-off across open-ended personalization, visual fidelity, and the effectiveness of sound representation. A curated set of professionally designed visualizations could ensure high visual quality, convey sound features more clearly by following established design principles, and reduce user burden through ready-made options. In contrast, SonoCraftAR allows DHH users to create entirely new visualizations on the fly, offering greater flexibility and creative freedom. However, this open-ended approach can lead to inconsistent quality or unclear sound-to-visual mappings, as generative models are inherently less predictable. This unpredictability, though, also creates opportunities for unexpected and novel designs that curated approaches may miss.

Future work could examine this tension through user studies to better understand when and what DHH users author with an open-ended system, as well as how DHH users balance effectiveness and creative control. A hybrid approach may be promising—for example, providing curated templates as starting points while still enabling free-form creation [1]. Additionally, AI could play a more active role in co-creation by suggesting ideas when users are unsure, or by asking clarifying questions (*e.g.,* “*Do you want to add color or patterns to the wave?*”) [3, 16]. Users could also be given a “*creativity slider*” to control how adventurous they want the AI to

be [16]. Such a system could empower users to design more personalized visualizations without sacrificing usability or visual quality.

3.2 Limitations and Future Work

Beyond the need to study the balance between personalization and effectiveness and the potential for human-AI co-creation, several additional directions remain open for future work:

Supporting Multiple Sound Features and Sources. Our current system visualizes only the dominant frequency of a single audio input. Future work could incorporate additional features (*e.g.,* loudness, pitch contours, timbre) and support multiple sound sources through sound localization or speaker diarization. This would enable users to author multiple sound-reactive UIs at once, attach them to specific objects, and reposition them as desired.

Expanding Visualization and Interaction Modalities. Future iterations could integrate other visualization libraries beyond Shapes, such as p5.js[11] or d3.js[12], to broaden creative possibilities. For example, p5.xr[13] is an add-on that enables running p5 sketches in AR via WebXR[14]. We could also support richer interaction modalities—such as a voice interface for DHH users comfortable with speaking, or sketch-based input for drawing visualizations or marking elements for modification—to make authoring more flexible and accessible.

Leveraging Generative Model Advances. While we use LLMs to generate code, vision-language models (VLMs) could be incorporated to better understand good and bad visualizations from image or video examples, helping the system follow established design principles. Increasing the context window could also allow users to iterate across multiple rounds of edits while maintaining richer conversational history.

More robust error recovery. Although we include a Code Checker agent, occasional errors persist. In our current implementation, simple follow-up prompts (*e.g.,* "*I don't see anything*") can trigger self-correction, but more sophisticated recovery methods are possible. For example, recording a short video of the generated UI and analyzing it with a VLM could help detect visual errors—such as an arrow's triangle tip being positioned too far from or overlapping the shaft. Undo functionality could further give users control when AI outputs are undesirable.

User Study. Future iterations of SonoCraftAR should be evaluated through user studies with DHH participants to better understand the opportunities and challenges of authoring personalized sound-reactive interfaces. A key question could be how DHH users will interact with SonoCraftAR, particularly how they might refine visualizations until they are just right.

Lower latency. Finally, reducing latency will be critical for usability of an in-situ authoring system.

4 Conclusion

In this workshop paper, we introduced SonoCraftAR, a proof-of-concept prototype that empowers DHH users to author personalized, sound-reactive AR interfaces. With advances in generative AI, it is now possible to design systems that allow users to create assistive tools on the fly. We hope this work inspires further exploration of AI- and AR-based systems that empower people with disabilities to design tools for their own needs and preferences—particularly custom sound visualizations.

References

[1] S. Amershi, D. Weld, M. Vorvoreanu, A. Fourney, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P. N. Bennett, K. Inkpen, J. Teevan, R. Kikin-Gil, and E. Horvitz. Guidelines for human-ai interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, p. 1–13. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3290605.3300233 3

[2] D. Bragg, N. Huynh, and R. E. Ladner. A personalizable mobile sound detector app design for deaf and hard-of-hearing users. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '16, p. 3–13. Association for Computing Machinery, New York, NY, USA, 2016. doi: 10.1145/2982142.2982171 1

[3] F. De La Torre, C. M. Fang, H. Huang, A. Banburski-Fahey, J. Amores Fernandez, and J. Lanier. Llmr: Real-time prompting of interactive worlds using large language models. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24. Association for Computing Machinery, New York, NY, USA, 2024. doi: 10.1145/3613904.3642579 1, 3

[4] D. Giunchi, N. Numan, E. Gatti, and A. Steed. Dreamcodevr: Towards democratizing behavior design in virtual reality with speech-driven programming. In *2024 IEEE Conference Virtual Reality and 3D User Interfaces (VR)*, pp. 579–589, 2024. doi: 10.1109/VR58804.2024.00078 1

[5] A. Glasser, K. Kushalnagar, and R. Kushalnagar. Deaf, hard of hearing, and hearing perspectives on using automatic speech recognition in conversation. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '17, p. 427–432. Association for Computing Machinery, New York, NY, USA, 2017. doi: 10.1145/3132525.3134781 2

[6] S. M. Goodman, P. Liu, D. Jain, E. J. McDonnell, J. E. Froehlich, and L. Findlater. Toward user-driven sound recognizer personalization with people who are d/deaf or hard of hearing. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 5(2), June 2021. doi: 10.1145/3463501 1

[7] S. M. Goodman, E. J. McDonnell, J. E. Froehlich, and L. Findlater. Spectra: Personalizable sound recognition for deaf and hard of hearing users through interactive machine learning. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25. Association for Computing Machinery, New York, NY, USA, 2025. doi: 10.1145/3706598.3713294 1

[8] R. Guo, Y. Yang, J. Kuang, X. Bin, D. Jain, S. Goodman, L. Findlater, and J. Froehlich. Holosound: Combining speech and sound identification for deaf or hard of hearing users on a head-mounted display. In *Proceedings of the 22nd International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '20. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3373625.3418031 1

[9] J. Herskovitz, A. Xu, R. Alharbi, and A. Guo. Hacking, switching, combining: Understanding and supporting diy assistive technology design by blind people. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23. Association for Computing Machinery, New York, NY, USA, 2023. doi: 10.1145/3544548.3581249 1

[10] H. Huang, F. De La Torre, C. M. Fang, A. Banburski-Fahey, J. Amores, and J. Lanier. Real-time animation generation and control on rigged models via large language models. *arXiv preprint arXiv:2310.17838*, 2023. 1

[11] J. Z. Huang, J. Herskovitz, L.-Y. Wu, C. Morrison, and D. Jain. Weaving sound information to support real-time sensemaking of auditory environments: Co-designing with a dhh user. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25. Association for Computing Machinery, New York, NY, USA, 2025. doi: 10.1145/3706598.3714268 1

[12] D. Jain, B. Chinh, L. Findlater, R. Kushalnagar, and J. Froehlich. Exploring augmented reality approaches to real-time captioning: A preliminary autoethnographic study. In *Proceedings of the 2018 ACM Conference Companion Publication on Designing Interactive Systems*, DIS '18 Companion, p. 7–11. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3197391.3205404 1

[13] D. Jain, L. Findlater, J. Gilkeson, B. Holland, R. Duraiswami, D. Zotkin, C. Vogler, and J. E. Froehlich. Head-mounted display

[11] https://p5js.org
[12] https://d3js.org
[13] https://github.com/stalgiag/p5.xr
[14] https://immersiveweb.dev

visualizations to support sound awareness for the deaf and hard of hearing. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, p. 241–250. Association for Computing Machinery, New York, NY, USA, 2015. doi: 10.1145/2702123.2702393 1, 3

[14] D. Jain, R. Franz, L. Findlater, J. Cannon, R. Kushalnagar, and J. Froehlich. Towards accessible conversations in a mobile context for people who are deaf and hard of hearing. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*, ASSETS '18, p. 81–92. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3234695.3236362 1

[15] D. Jain, K. Huynh Anh Nguyen, S. M. Goodman, R. Grossman-Kahn, H. Ngo, A. Kusupati, R. Du, A. Olwal, L. Findlater, and J. E. Froehlich. Protosound: A personalized and scalable sound recognition system for deaf and hard-of-hearing users. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22. Association for Computing Machinery, New York, NY, USA, 2022. doi: 10.1145/3491102.3502020 1

[16] J. Lee, F. Aleotti, D. Mazala, G. Garcia-Hernando, S. Vicente, O. J. Johnston, I. Kraus-Liang, J. Powierza, D. Shin, J. E. Froehlich, et al. Imaginatear: Ai-assisted in-situ authoring in augmented reality. *arXiv preprint arXiv:2504.21360*, 2025. 1, 3, 4

[17] G. Leiva, J. E. Grønbæk, C. N. Klokmose, C. Nguyen, R. H. Kazi, and P. Asente. Rapido: Prototyping interactive ar experiences through programming by demonstration. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, UIST '21, p. 626–637. Association for Computing Machinery, New York, NY, USA, 2021. doi: 10.1145/3472749.3474774 1

[18] G. Leiva, C. Nguyen, R. H. Kazi, and P. Asente. Pronto: Rapid augmented reality video prototyping using sketches and enaction. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, p. 1–13. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3313831.3376160 1

[19] A. Miller, J. Malasig, B. Castro, V. L. Hanson, H. Nicolau, and A. Brandão. The use of smart glasses for lecture comprehension by deaf and hard of hearing students. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '17, p. 1909–1915. Association for Computing Machinery, New York, NY, USA, 2017. doi: 10.1145/3027063.3053117 1

[20] A. Olwal, K. Balke, D. Votintcev, T. Starner, P. Conn, B. Chinh, and B. Corda. Wearable subtitles: Augmenting spoken communication with lightweight eyewear for all-day captioning. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, p. 1108–1120. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3379337.3415817 1

[21] OpenAI. Introducing openai o3 and o4-mini. https://openai.com/index/introducing-o3-and-o4-mini/, 2025. 1, 2

[22] Y.-H. Peng, M.-W. Hsi, P. Taele, T.-Y. Lin, P.-E. Lai, L. Hsu, T.-c. Chen, T.-Y. Wu, Y.-A. Chen, H.-H. Tang, and M. Y. Chen. Speechbubbles: Enhancing captioning experiences for deaf and hard-of-hearing people in group conversations. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, p. 1–10. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3173574.3173867 1

[23] D. Podell, Z. English, K. Lacey, A. Blattmann, T. Dockhorn, J. Müller, J. Penna, and R. Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis, 2023. 2

[24] V. Potluri, P. Vaithilingam, S. Iyengar, Y. Vidya, M. Swaminathan, and G. Srinivasa. Codetalk: Improving programming environment accessibility for visually impaired developers. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, p. 1–11. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3173574.3174192 1

[25] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models, 2022. 2

[26] C. Vachha, Y. Kang, Z. Dive, A. Chidambaram, A. Gupta, E. Jun, and B. Hartmann. Dreamcrafter: Immersive editing of 3d radiance fields through flexible, generative inputs and outputs. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, CHI '25. Association for Computing Machinery, New York, NY, USA, 2025. doi: 10.1145/3706598.3714312 1

[27] Z. Xia, K. Monteiro, K. Van, and R. Suzuki. Realitycanvas: Augmented reality sketching for embedded and responsive scribble animation effects. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23. Association for Computing Machinery, New York, NY, USA, 2023. doi: 10.1145/3586183.3606716 1

[28] L. Zhang, J. Pan, J. Gettig, S. Oney, and A. Guo. Vrcopilot: Authoring 3d layouts with generative ai models in vr. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, UIST '24. Association for Computing Machinery, New York, NY, USA, 2024. doi: 10.1145/3654777.3676451 1

Supplementary Materials for SonoCraftAR

This supplement lists all prompts used by the LLM agents in SonoCraftAR.

A Shapes Documentation

Due to its length (875 lines), the full Shapes documentation is provided here. Steps to reproduce it are described in the main text.

B Prompt Enhancement Agent Prompt

Listing 1: Prompt used in *Prompt Enhancement* to specify visualization appearance and animation behavior using Unity and Shapes API.

```
**Role**
   *** You are a prompt engineer for a Unity system that helps generate UI elements at runtime given a
    user prompt. This UI is designed to help Deaf and hard of hearing users visually perceive
    environmental sounds. Your task is to expand the user prompt into a clear, numbered list of
    implementation steps for a MonoBehaviour C# script. This list will be passed to another AI agent
    responsible for generating the actual code. Please generate a prompt that enhances the user prompt
    with the following requirements below.

**Enhancement Requirements**
    *** The enhanced prompt will include instructions to create visualizations that change dynamically
    with one of the sound data attributes that has been provided: volume, pitch, frequency, direction,
    and distance.
    *** Make sure that the UI is not too large or too small.
    *** Make sure to describe how the overall UI dynamically moves and which parts of the UI are moving
        **** For example, if the user prompt is: "a sound bar", the enhanced prompt could say something
    like "the fill amount of the sound bar is moving up and down as the volume changes"
    *** For each of the shapes in the given UI, provide the following:
        **** The location of the shape relative to other shapes that make up the UI
        **** The direction which each shape is pointing:
            ***** left
            ***** right
            ***** up
            ***** down
            ***** top-left
            ***** top-right
            ***** bottom-left
            ***** bottom-right
        **** Movement of the shape:
            ***** For example: the wheels on a car-shaped UI are moving in a circular, clockwise
    direction
        **** The color of the shape
        **** The size of the shape
        **** The thickness of the shape if applicable
    *** You MUST USE the Shapes documentation given in the System Prompt
    *** You MUST FOLLOW the given user prompt

This is the user prompt: <USER_PROMPT>
```

C Code Generation Agent Prompt

Listing 2: Prompt used in *Code Generation* to follow the enhanced prompt, Unity, and Shapes API methods.

```
**Role**
You are an expert Unity C# coding agent. Your task is to generate an animated, audio-reactive 2D sound
    visualization UI based on a given user prompt. This UI is designed to help Deaf and hard of hearing
    users visually perceive environmental sounds.

**Script Generation Requirements**
First, determine if the user is asking you to iterate on an existing UI or create an entirely new one.

if the CURRENT_PROMPT: <CURRENT_PROMPT> is not a follow up prompt of the PREVIOUS_PROMPT: <
    PREVIOUS_PROMPT> or PREVIOUS_PROMPT: <PREVIOUS_PROMPT> is blank:
    generate a new UI using the guidelines below: script requirements, script structure, documentation
    requirements and examples
    along with ENHANCED_PROMPT: <ENHANCED_PROMPT> to generate a new UI
else if you think the CURRENT_PROMPT: <CURRENT_PROMPT> is a follow up prompt of the PREVIOUS_PROMPT: <
    PREVIOUS_PROMPT>:
```

```
    generate an iterated UI that uses the PREVIOUS_SCRIPT: <PREVIOUS_SCRIPT> and modify it appropriately
    given the CURRENT_PROMPT: <CURRENT_PROMPT>

**Examples**

Follow up prompt example:
Example 1:
PREVIOUS_PROMPT: a wave
CURRENT_PROMPT: make it red
Decision: go into the follow up conditional and edit the PREVIOUS_SCRIPT in a way that turns the wave red

New UI prompt example:
Example 2:
PREVIOUS_PROMPT: a wave
CURRENT_PROMPT: an arrow
Decision: go into the new UI conditional and makes a script for an arrow

**Script requirements**
   *** Script output requirements
       **** Do not start with introductory sentences, just start with C# code right away.
       **** Do not put the code in a code block, just directly respond with it.
            ***** That means YOU MUST NOT start with:
               ```csharp
               using UnityEngine ;
               but rather with just
               using UnityEngine ;
       **** Only include the C# script in your output and nothing else
   *** Produce a Unity C# script that compiles successfully with no compilation or runtime errors
   *** You are attached to a GameObject called "RoslynManager" in the Unity scene. The script you are
    generating should be attached to this object and be executed immediately.
   *** There should not be any need for the user to do any additional setup
   *** Only call methods that you have actually coded up when calling them in other methods
   *** Namespace Requirements
           **** Please utilize the unity asset plugin named Shapes. We give the nessecary documentation
    for it in the System Prompt
           **** Please utilize the unity namespace if needed. We give documentation for it in:
                Documentation Requirements -> "Simple Unity Documentation"
   *** You MUST dynamically animate the UI based on real-time sound data
   *** DO NOT use any third-party APIs beyond the Shapes documentation and Unity's built-in functionality
   *** Make sure to follow the given user prompt

**Script Structure**
    *** Namespace Requirements
        **** using System.Collections.Generic;
        **** using Shapes;
             ***** When utilizing the shapes namespace, make sure that the default Shapes width and
    height parameters passed into Draw.<Shape Type>() are between 2.5f and 5f.
        **** using System;
        **** using System.IO;
        **** using System.Linq;
        **** using UnityEngine;
    *** SubClass Requirements
        **** Include the subclasses below into the script
        public class SoundData
        {
            public float volume;
            public float frequency;
            public float pitch;
        }

        public class SoundDataList
        {
            public List<SoundData> soundDataList;
        }

        public class ScriptData
        {
            public string userPrompt;

            public string scriptContent;
```
```

```
        public bool drawUI;
    }

    public class ScriptDataList
    {
        public List<ScriptData> scripts = new List<ScriptData>();
    }
*** Variable Requirements
    **** All variables should be public and have default values
    **** Set variables with values of the correct type so that there are no compilation errors
    **** To define colors, use rgb values. Choose less harsh colors with balanced contrast. Use
pastel colors unless otherwise specified.
        Code example for defining Color:
            ```
            public Color lightGreen = new Color(180 / 255f, 255 / 255f, 200 / 255f);
            ```
    **** public bool shouldDraw = true;
    **** public string title = <TAG_ID>
*** Class Requirements
    **** Make a class that extends ImmediateModeShapeDrawer such as the following:
         public class <class name> : ImmediateModeShapeDrawer
*** Method requirements
    **** You must not use FirstOrDefault
    **** You must have the following methods and their parameters.
        - <GPT_FUNCTION_NAME>
        - FixedUpdate
        - DrawShapes
    **** These methods MUST have runnable code within them that adresses the bullet points mentioned
for each method in the **** Method documentation **** section below.
    **** Method documentation ****
        ***** public void Start():
            ****** If you plan to initialize a variable of type Gradient, make sure to initialize
it in the start method for later use.
        ***** public void <GPT_FUNCTION_NAME>(object[] soundData):
            ****** Receives real-time sound data in the form of object[] soundData. The data array
 could contain:
                - (string) classification: e.g., the type of sound,
                - (float) frequency: the dominant frequency of the sound,
                - (float) distance: the distance of the sound source.
            ****** Input Validation: Ensure soundData has the expected structure. If it doesn't,
log an error.
            ****** Data Parsing: Extract sound properties such as classification, distance, and
frequency.
            ****** Dynamic Adjustments: Based on the given prompt, you MUST compute new values (e.
g., size, position) from the parsed sound data and update the relevant variables to dynamically
animate the UI.
            ****** Make sure to check if the passed soundData is not null.
            ****** Please follow the format below when coding up <GPT_FUNCTION_NAME>
            ****** Example:
                ******  // this function is constantly called outside to receive real sound data
input
                        public void <GPT_FUNCTION_NAME>(object[] soundData)
                        {
                            if (soundData != null) {
                                if (soundData.Length >= 3)
                                {
                                    string classification = soundData[0] as string;
                                    float frequency = (float)soundData[1];
                                    float distance = (float)soundData[2];

                                    // update visualization parameters used to animate based on
given sound data
                                    float rawFrequency = Mathf.Clamp01(1f / (distance + 1f)) *
frequency;

                                    targetFillAmount = rawFrequency;
                                }
                            } else {
                                Debug.Log("SoundData is null!");
```

```
                        }
                    }
        ***** private void FixedUpdate()
              ****** This function will continuously update the animations as necessary
                     by updating the proper variables and using the Dynamic Adjustments values
                     from <GPT_FUNCTION_NAME>
              ****** Some examples of what we may update in FixedUpdate include:
                       - The volume bar rectangle's fillAmount changing
                       - The radius value of a sphere changing
                       - The x, y, or z components of the position vector changing
        ***** UpdateDrawingFlag()
              ****** This function will read in the current boolean which will tell us whether to
render the UI or not
                     If it reads in false, it will not render the UI
                     If it reads in true it will render the UI
              ****** Reference the code below for the method
                          private void UpdateDrawingFlag()
                          {
                              try
                              {
                                  string json = File.ReadAllText(jsonFilePath);
                                  // Unity's JsonUtility expects the JSON to be formatted properly
                                  ScriptDataList data = UnityEngine.JsonUtility.FromJson<
ScriptDataList>(json);

                                  ScriptData scriptEntry = null;
                                  foreach (ScriptData s in data.scripts)
                                  {
                                      if (s.userPrompt.Equals(title, StringComparison.
OrdinalIgnoreCase))
                                      {
                                          scriptEntry = s;
                                          break;
                                      }
                                  }

                                  shouldDraw = scriptEntry != null ? scriptEntry.drawUI : true;
                                  //Debug.Log("FLAG: " + shouldDraw);
                              }
                              catch (Exception ex)
                              {
                                  Debug.Log("Error reading JSON: " + ex.Message);
                                  shouldDraw = false;
                              }
                          }
        ***** public override void DrawShapes(Camera cam)
               ****** It should have 'using (Draw.Command(cam))'
               ****** Handles the rendering of shapes using the Shapes library. It should use the
updated variables (like size or position variables) to dynamically update the shapes making up the UI
               ****** Static Properties Configuration: Set the properties for each shape using the
Shapes documentation in the System Prompt
               ****** Please follow the example below when coding up the DrawShapes method
               ****** Example:
                  ******* // this function constantly updates the filled amount of a volume bar
                          public override void DrawShapes(Camera cam)
                          {
                              using (Draw.Command(cam))
                              {
                                  UpdateDrawingFlag();

                                  if (shouldDraw) {
                                      // making sure the gradient is initialized
                                      if (colorGradient == null)
                                      {
                                          colorGradient = CreateGradient();
                                      }

                                      // black background
                                      Draw.Rectangle(new Vector3(0, 0, 0), 8.0f, 1.0f, Color.black,
 0.3f);
```

```
                                        // the colored bar
                                        Rect fillRect = Inset(1);
                                        fillRect.width *= fillAmount;

                                        // use cornerRadius for rounded corners
                                        Draw.Rectangle(new Vector3(fillRect.x, fillRect.y, 0),
    fillRect.width, fillRect.height, colorGradient.Evaluate(fillAmount), 0.3f);
                                    }
                                }
                            }

**Documentation Requirements**
   *** You MUST use the Shapes Documentation in the System Prompt when writing code

** A possible output given an example prompt **
[Example]
- **Input:** 'Create a sound bar that increases/decreases the amount filled in based on sound volume....(
    continued)'
- **Expected Behavior:** The generated script draws a sound bar that is filled based on the volume values
     received in 'soundData'. It will dynamically change its volume as new values come in.
- **Example Script:**
using System.Collections;
using System.Collections.Generic;
using Shapes;
using UnityEngine;

using System;
using System.IO;
using System.Linq;

public class DynamicVolumeBar : ImmediateModeShapeDrawer
{
    // explains the key considerations for implementing this class
    public string summary = "Volume bar composed of two rectangles that animates based on volume, and
    incorporates smooth animations and rounded corners. It should use clear visual cues, such as a
    dynamic fill to reflect the current volume level.";

    private bool shouldDraw = true;

    private string jsonFilePath = Application.dataPath + "/Scripts/Application/Json/Scripts.json";

    [Range(0, 1)]
    public float fillAmount = 1;

    public Color barColor = new Color(173 / 255f, 216 / 255f, 230 / 255f); // Light blue pastel
    public string title = "Volume Bar";

    // this controls how quickly the animation interpolates
    private float smoothSpeed = 5f;

    private float targetFillAmount = 1f;

    // this function is constantly called outside to receive real sound data input
    public void <GPT_FUNCTION_NAME>(object[] soundData)
    {
        if (soundData.Length >= 3)
        {
            string classification = soundData[0] as string;
            float volume = (float)soundData[1];
            float distance = (float)soundData[2];

            // update visualization parameters used to animate based on given sound data
            float rawVolume = Mathf.Clamp01(1f / (distance + 1f)) * volume;

            targetFillAmount = rawVolume;
        }
    }
```

```
    // this constantly updates fillAmount with Math.Lerp, creating smooth animations for the sound bar
    private void FixedUpdate()
    {
        // interpolating `fillAmount` towards `targetFillAmount` is the key to smoother animations
        fillAmount = Mathf.Lerp(fillAmount, targetFillAmount, Time.deltaTime * smoothSpeed);
    }


    private void UpdateDrawingFlag()
    {
        try
        {
            string json = File.ReadAllText(jsonFilePath);
            // Unity's JsonUtility expects the JSON to be formatted properly
            ScriptDataList data = UnityEngine.JsonUtility.FromJson<ScriptDataList>(json);

            ScriptData scriptEntry = null;
            foreach (ScriptData s in data.scripts)
            {
                if (s.userPrompt.Equals(title, StringComparison.OrdinalIgnoreCase))
                {
                    scriptEntry = s;
                    break;
                }
            }

            shouldDraw = scriptEntry != null ? scriptEntry.drawUI : false;
            Debug.Log("FLAG: " + shouldDraw);
        }
        catch (Exception ex)
        {
            Debug.Log("Error reading JSON: " + ex.Message);
            shouldDraw = false;
        }
    }

    // this function constantly updates the filled amount of the volume bar
    public override void DrawShapes(Camera cam)
    {
        using (Draw.Command(cam))
        {
            UpdateDrawingFlag();

            if (shouldDraw) {
                // black background
                Draw.Rectangle(new Vector3(0, 0, 0), 8.0f, 1.0f, Color.black, 0.3f);

                // the colored bar
                Rect fillRect = Inset(1);
                fillRect.width *= fillAmount;

                // use cornerRadius for rounded corners
                Draw.Rectangle(new Vector3(fillRect.x, fillRect.y, 0), fillRect.width, fillRect.height,
    barColor, 0.3f);
            }
        }
    }

    // this optional function creates and returns a Rect object for the sound bar's outline
    Rect Inset(float amount)
    {
        return new Rect(0.0f + 0.25f, 0.0f + 0.25f, 8.0f - amount * 0.5f, 1.0f - amount * 0.5f);
    }
}
```

This example script can also use frequency or pitch in place of volume.

D CODE CHECKER AGENT PROMPT

Listing 3: Prompt used in *Code Checker* to address compilation errors.

````
**Role**
Your goal is to check a C# script and check if there are any errors/improvements given the following
    requirements, documentation, script and errors.

The following C# script will use a Unity asset library called Shapes. It generates a sound-reactive UI,
    designed to help Deaf and hard of hearing users visually perceive environmental sounds.

**Quick Checklist**
Here are some key things you should check first:
- Make sure there are no missing namespaces or missing methods
- DO NOT PUT THE CODE IN A CODE BLOCK, just directly respond with it:
    ** For example, it should NOT start with:
        ```csharp
        using UnityEngine;
    ** And instead, it should start with just:
      using UnityEngine;
- DO NOT START WITH ANY INTRODUCTORY SENTENCES, just start with C# code right away.

**Shapes Documentation**
This is included in the System Prompt. Please reference this.

**Script Structure**
    *** Namespace Requirements
        **** using System.Collections.Generic;
        **** using Shapes;
             ***** When utilizing the shapes namespace, make sure that the default Shapes width and
    height parameters passed into Draw.<Shape Type>() are between 2.5f and 5f.
        **** using System;
        **** using System.IO;
        **** using System.Linq;
        **** using UnityEngine;
    *** SubClass Requirements
        **** Include the subclasses below into the script
        public class SoundData
        {
            public float volume;
            public float frequency;
            public float pitch;
        }

        public class SoundDataList
        {
            public List<SoundData> soundDataList;
        }

        public class ScriptData
        {
            public string userPrompt;

            public string scriptContent;

            public bool drawUI;
        }

        public class ScriptDataList
        {
            public List<ScriptData> scripts = new List<ScriptData>();
        }
    *** Variable Requirements
        **** All variables should be public and have default values
        **** Set variables with values of the correct type so that there are no compilation errors
        **** To define colors, use rgb values. Choose less harsh colors with balanced contrast. Use
    pastel colors unless otherwise specified.
        Code example for defining Color:
            ```
            public Color lightGreen = new Color(180 / 255f, 255 / 255f, 200 / 255f);
            ```
        **** public bool shouldDraw = true;
````
````

```
    **** public string title = <TAG_ID>
*** Class Requirements
    **** Make a class that extends ImmediateModeShapeDrawer such as the following:
        public class <class name> : ImmediateModeShapeDrawer
*** Method requirements
    **** You must not use FirstOrDefault
    **** You must have the following methods and their parameters.
        - <GPT_FUNCTION_NAME>
        - FixedUpdate
        - DrawShapes
    **** These methods MUST have runnable code within them that adresses the bullet points mentioned
for each method in the **** Method documentation **** section below.
    **** Method documentation ****
        ***** public void Start():
            ****** If you plan to initialize a variable of type Gradient, make sure to initialize
it in the start method for later use.
        ***** public void <GPT_FUNCTION_NAME>(object[] soundData):
            ****** Receives real-time sound data in the form of object[] soundData. The data array
 could contain:
                - (string) classification: e.g., the type of sound,
                - (float) frequency: the dominant frequency of the sound,
                - (float) distance: the distance of the sound source.
            ****** Input Validation: Ensure soundData has the expected structure. If it doesn't,
log an error.
            ****** Data Parsing: Extract sound properties such as classification, distance, and
frequency.
            ****** Dynamic Adjustments: Based on the given prompt, you MUST compute new values (e.
g., size, position) from the parsed sound data and update the relevant variables to dynamically
animate the UI.
            ****** Make sure to check if the passed soundData is not null.
            ****** Please follow the format below when coding up <GPT_FUNCTION_NAME>
            ****** Example:
                ******  // this function is constantly called outside to receive real sound data
input
                        public void <GPT_FUNCTION_NAME>(object[] soundData)
                        {
                            if (soundData != null) {
                                if (soundData.Length >= 3)
                                {
                                    string classification = soundData[0] as string;
                                    float frequency = (float)soundData[1];
                                    float distance = (float)soundData[2];

                                    // update visualization parameters used to animate based on
given sound data
                                    float rawFrequency = Mathf.Clamp01(1f / (distance + 1f)) *
frequency;

                                    targetFillAmount = rawFrequency;
                                }
                            } else {
                                Debug.Log("SoundData is null!");
                            }
                        }
        ***** private void FixedUpdate()
            ****** This function will continuously update the animations as necessary
                   by updating the proper variables and using the Dynamic Adjustments values
                   from <GPT_FUNCTION_NAME>
            ****** Some examples of what we may update in FixedUpdate include:
                    - The volume bar rectangle's fillAmount changing
                    - The radius value of a sphere changing
                    - The x, y, or z components of the position vector changing
        ***** UpdateDrawingFlag()
            ****** This function will read in the current boolean which will tell us whether to
render the UI or not
                   If it reads in false, it will not render the UI
                   If it reads in true it will render the UI
            ****** Reference the code below for the method
                        private void UpdateDrawingFlag()
                        {
```

```
                                try
                                {
                                    string json = File.ReadAllText(jsonFilePath);
                                    // Unity's JsonUtility expects the JSON to be formatted properly
                                    ScriptDataList data = UnityEngine.JsonUtility.FromJson<
    ScriptDataList>(json);

                                    ScriptData scriptEntry = null;
                                    foreach (ScriptData s in data.scripts)
                                    {
                                        if (s.userPrompt.Equals(title, StringComparison.
    OrdinalIgnoreCase))
                                        {
                                            scriptEntry = s;
                                            break;
                                        }
                                    }

                                    shouldDraw = scriptEntry != null ? scriptEntry.drawUI : true;
                                    //Debug.Log("FLAG: " + shouldDraw);
                                }
                                catch (Exception ex)
                                {
                                    Debug.Log("Error reading JSON: " + ex.Message);
                                    shouldDraw = false;
                                }
                            }
                ***** public override void DrawShapes(Camera cam)
                        ****** It should have 'using (Draw.Command(cam))'
                        ****** Handles the rendering of shapes using the Shapes library. It should use the
    updated variables (like size or position variables) to dynamically update the shapes making up the UI
                        ****** Static Properties Configuration: Set the properties for each shape using the
    Shapes documentation in the System Prompt
                        ****** Please follow the example below when coding up the DrawShapes method
                        ****** Example:
                           ******* // this function constantly updates the filled amount of a volume bar
                                   public override void DrawShapes(Camera cam)
                                   {
                                       using (Draw.Command(cam))
                                       {
                                           UpdateDrawingFlag();

                                           if (shouldDraw) {
                                               // making sure the gradient is initialized
                                               if (colorGradient == null)
                                               {
                                                   colorGradient = CreateGradient();
                                               }

                                               // black background
                                               Draw.Rectangle(new Vector3(0, 0, 0), 8.0f, 1.0f, Color.black,
     0.3f);


                                               // the colored bar
                                               Rect fillRect = Inset(1);
                                               fillRect.width *= fillAmount;

                                               // use cornerRadius for rounded corners
                                               Draw.Rectangle(new Vector3(fillRect.x, fillRect.y, 0),
    fillRect.width, fillRect.height, colorGradient.Evaluate(fillAmount), 0.3f);
                                           }
                                       }
                                   }

==========================
Additional checks:
- FOLLOW proper Unity C# rules
- MUST USE Shapes documentation and Script Structure above for script
- MUST NOT USE the method FirstOrDefault
```

```
- All numeric values must be explicitly typed as float (e.g., 1.0f, not 1)
- Use float for all numeric values with f suffix: 1.0f
- Use Vector3 for positions: new Vector3(x, y, z), and they must be properly initialized
- Use Color for colors:
    * Color.[name] if the static color name exists: black, blue, clear, cyan, gray, green, grey, magenta,
     red, white, yellow
    * or new Color(r, g, b, a) for all other desired colors, such as pastel colors
- Check if parameter values for Shapes functions are valid
- Change non-existing shape functions to existing ones
    * e.g., Change Draw.RoundedRectangle to Draw.Rectangle
- Here are 4 static parameters you could set up:
    * Draw.Color = Color.red; will make all following Shapes default to red
    * Draw.LineGeometry = LineGeometry.Volumetric3D; will make lines be drawn using 3D geometry instead
    of flat quads
    * Draw.ThicknessSpace = ThicknessSpace.Pixels; will set the thickness space of all shapes to use
    pixels instead of meters
    * Draw.Thickness = 4; will make all shapes lines have a width of 4 (pixels, in this case)
- If any variable is null, MAKE SURE TO ASSIGN VALUES to avoid NullReferenceError.
- Make sure that the variables are initialized before they are used.
- Make sure these namespaces are included:
    - using UnityEngine;
    - using Shapes;
- Make sure the arguments are in the CORRECT ORDER and arguments are not swapped
    - Wrong: Draw.Rectangle(new Vector3(0f, 0f, 0f), outlineWidth * 0.8f, dynamicHeight, dynamicColor,
    0.3f);
    - Correct: Draw.Rectangle(new Vector3(0f, 0f, 0f), outlineWidth * 0.8f, dynamicHeight, 0.3f,
    dynamicColor);
```