

특집논문 (Special Paper)

방송공학회논문지 제31권 제5호, 2026년 9월 (JBE Vol.31, No.5, September 2026)

<https://doi.org/10.5909/JBE.2026.31.5.906>

ISSN 2287-9137 (Online) ISSN 1226-7953 (Print)

라이브 코딩 퍼포먼스를 위한 생성형 AI 기반 오디오비주얼 시스템 설계 및 구현

정 지 웅^{a)}, 김 한 성^{b)}, 신 춘 성^{a)†}

Design and Implementation of a Generative AI-Based Audiovisual System for Live Coding Performance

Jiung Jeong^{a)}, Hansung Kim^{b)}, and Choonsung Shin^{a)†}

요 약

본 논문은 라이브 코딩 퍼포먼스를 쉽게 지원하고 연주자의 의도 반영을 높이기 위한 생성형 AI 기반 오디오비주얼 시스템을 제안한다. 제안한 시스템은 연주자의 음성 명령과 현재 공연의 오디오비주얼 출력, 실행 중인 소스 코드를 멀티모달 생성형 AI에 한 번에 입력하여 사운드(SuperCollider)와 셰이더 영상(GLSL) 코드를 동시에 생성하고 그 결과를 즉시 공연에 반영하도록 설계하였다. 제안한 시스템은 TouchDesigner 렌더러와 SuperCollider 사운드 서버, 파이썬 제어 서버를 활용하였으며 브라우저 기반 공연용 콘솔 형태로 구현되었다. 제안한 시스템의 성능평가를 위해 절대적·상대적·음악적 특성을 갖는 30개의 명령을 구성하고, Gemini 3.5 Flash를 기준 모델로 Gemini 3.1 Flash-Lite를 비교 대상인 경량 모델로 설정하여 오디오비주얼 생성결과를 분석하였다. 분석 결과, 기준 모델은 경량 모델보다 응답 시간이 다소 길었으나 의도 반영률은 더 높았으며, 음성 직접 입력과 오디오비주얼 출력 피드백이 최종 생성 결과의 의도 반영률을 향상시키는 것으로 나타났다.

Abstract

This paper proposes a generative AI-based audiovisual system designed to facilitate live coding performance and enhance the reflection of the performer's intent. The proposed system feeds the performer's voice command, the current audiovisual output of the performance, and the running source code into a multimodal generative AI in a single request, generating sound (SuperCollider) and shader visual (GLSL) code simultaneously and applying the results to the performance immediately. The system is implemented with a TouchDesigner real-time renderer, a SuperCollider sound server, a Python control server, and a browser-based performance console. To evaluate the performance of the proposed system, we constructed a set of 30 commands incorporating absolute, relative, and musical characteristics. Gemini 3.5 Flash was selected as the baseline model, while Gemini 3.1 Flash-Lite was used as the lightweight comparison model. The results showed that although the baseline model had a slightly longer response time than the lightweight model, it achieved a higher intent alignment rate. Direct voice input and audiovisual output feedback also improved the intent alignment of the final generated outputs.

Keyword : Live coding, Multimodal generative AI, Audiovisual performance, Automatic code generation, Voice

Copyright © 2026 Korean Institute of Broadcast and Media Engineers. All rights reserved.

“This is an Open-Access article distributed under the terms of the Creative Commons BY-NC-ND (<http://creativecommons.org/licenses/by-nc-nd/3.0>) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited and not altered.”

1. 서론

라이브 코딩 퍼포먼스(Live coding performance)는 음원이나 영상을 재생하는 대신, 연주자가 공연 중에 직접 코드를 작성하고 수정하면서 사운드와 영상을 실시간으로 만들어 가는 퍼포먼스 형식이다^[1]. 이러한 퍼포먼스는 코드가 곧 악기가 되는 형식의 인지적 특성, 즉 프로그래밍 언어를 악기처럼 연주한다는 것이 어떤 부담과 표현력을 갖는지에 관한 연구로 이어지고 있다^[2].

라이브 코딩 퍼포먼스는 사운드 영역과 영상 영역을 중심으로 발전해 왔다. 사운드 영역에서는 실시간 사운드 합성을 위한 전용 언어이자 엔진인 SuperCollider^[3]와 그 위에서 리듬 패턴 조작에 특화된 TidalCycles^[4]가 널리 쓰인다. 영상 영역에서는 화면의 각 픽셀 색을 GPU에서 계산하는 셰이더(Shader) 프로그램(대표적으로 OpenGL의 셰이더 언어인 GLSL)을 실시간으로 작성하고 실행하는 Hydra^[5] 등의 환경이 자리 잡았다. 더 나아가 두 영역이 통합되는 오디오비주얼(audiovisual, AV) 퍼포먼스가 대두되고 있다. AV 시스템은 청각과 시각 표현을 긴밀하게 결합하여 어느 한쪽만으로는 얻기 어려운 몰입감과 표현력을 제공한다^[6]. 하지만 AV 퍼포먼스에서는 새로운 문제가 발생한다. 연주자는 사운드 합성 코드와 셰이더 코드라는 성격이 전혀 다른 두 언어를 무대 위에서 번갈아 작성해야 하므로 높은 프로그래밍 숙련도가 요구된다. 또한 공연 중에 영상이나

사운드의 일부 코드를 수정하는 동안 다른 부분의 음악적 흐름을 놓치게 되면서 인지 부하가 높아져 공연의 표현 흐름이 끊기게 된다.

최근에는 대규모 언어 모델(Large language model, LLM)을 활용해 자연어 프롬프트로부터 실행되는 코드 생성이 가능해짐에 따라^[7], 이를 창작 도구와 결합하려는 시도가 이어지고 있다. 대표적으로 GPT-4로 GLSL 셰이더를 대화형으로 발전시키는 AI Co-Artist^[8], 자연어 지시로 증강현실 화면의 셰이더를 생성하는 ShadAR^[9]가 제안되었다. 그러나 기존 연구는 세 가지 한계를 지닌다. 첫째, 입력이 텍스트 프롬프트나 마우스 조작에 머물러 있어, 코드 작성을 자동화하더라도 명령을 입력하는 동안 공연의 표현 흐름이 끊기는 문제는 그대로 남는다. 둘째, 사운드 또는 영상 중 한 도메인만 다루므로 AV 퍼포먼스의 이중 부담을 해소하지 못한다. 셋째, 모델이 현재의 AV 출력을 알지 못한 채 코드를 생성하므로, “지금보다 어렵게”처럼 현재 출력을 기준으로 삼는 상대적 명령을 해석할 수 없다. 또한 사운드 코드 생성에서 모델의 음악적 이해가 텍스트 표현만으로는 제한적이므로 음악적 의도를 텍스트가 아닌 소리 그대로 모델에 전달하는 설계가 필요하다^[10].

본 논문은 이러한 한계점을 해결하기 위해, 연주자의 음성만으로 사운드와 영상 코드를 동시에 생성하여 공연에 반영하는 생성형 AI 기반 AV 라이브 코딩 시스템을 제안한다. 기존 방식에서 연주자는 두 편집기를 오가며 성격이 다른 두 언어를 직접 타이핑하고 두 도메인이 맥락을 공유하지 않지만, 제안한 방식에서는 입력이 음성 하나로 통합되고 두 코드가 한 번의 호출로 함께 생성된다. 그림 1은 기존 AV 라이브 코딩 방식(그림 1 (a))과 제안 방식(그림 1 (b))의 관계를 비교하여 나타내고 있다. 제안한 시스템은 연주자의 음성 명령, 현재 공연 출력 화면과 사운드를 다시 받는 AV 출력 피드백, 실행 중인 소스 코드를 오디오와 비디오를 함께 이해하는 멀티모달 생성형 AI인 Gemini^[11]에 한 번의 요청으로 입력하도록 구성되었다. 또한 음성을 텍스트로 변환하지 않고 소리 그대로 입력하기 때문에 허밍이나 박수처럼 텍스트 변환 과정에서 사라지는 음악적 정보까지 명령으로 활용할 수 있는 장점이 있다. 생성형 AI 모델이 사운드와 영상 코드를 동시에 생성하면, 시스템은 이를 검증한 뒤 즉시 공연에 반영한다.

a) 전남대학교(Graduate School of Culture, Chonnam National University)
b) 사우스햄튼대학교(School of Electronics & Computer Science, University of Southampton)

✉ Corresponding Author : 신춘성(Choonsung Shin)

E-mail: cshin@jnu.ac.kr

Tel: +82-62-530-4092

ORCID: <https://orcid.org/0000-0003-2384-4022>

※ This work was supported by the Institute of Information & Communications Technology Planning & Evaluation(IITP)-Innovative Human Resource Development for Local Intellectualization program grant funded by the Korea government (MSIT) (IITP-2026-RS-2022-00156287, 50%) and by Culture, Sports and Tourism R&D Program through the Korea Creative Content Agency grant funded by the Ministry of Culture, Sports and Tourism in 2026 (Project Name: Content Illegal Streaming Infringement Analysis Digital Forensics Human Resource Development Center, Project Number: RS-2026-25513329, Contribution Rate: 50%)

· Manuscript received July 20, 2026; Revised August 26, 2026; Accepted August 26, 2026.

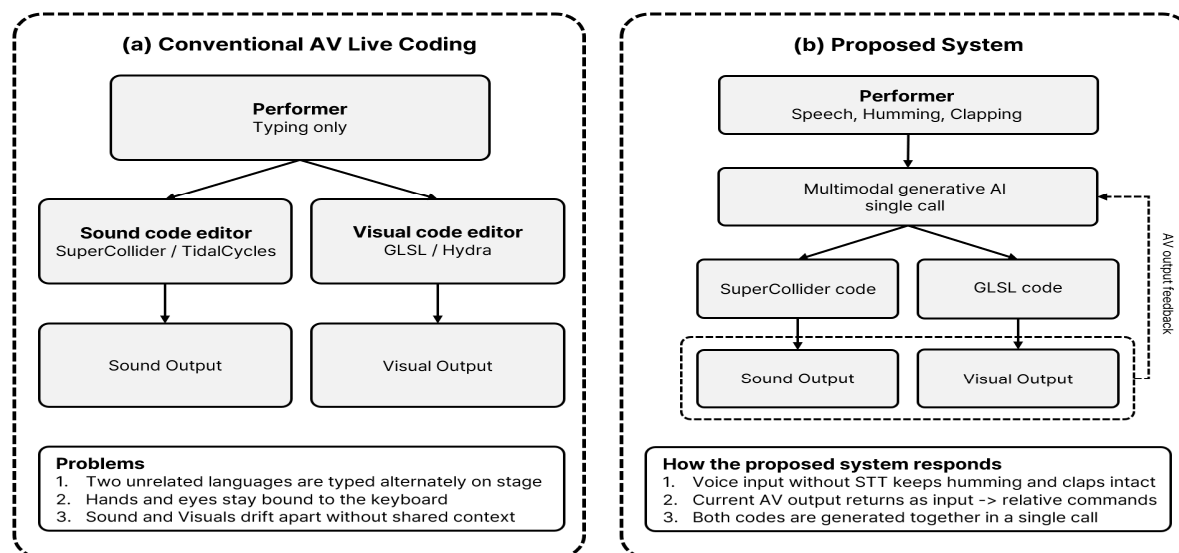


그림 1. 기존 AV 라이브 코딩 방식과 제안 시스템의 개념 비교

Fig. 1. Conceptual comparison between conventional AV live coding and the proposed system

본 논문의 기여는 다음과 같다. 첫째, 음성 명령과 AV 출력 피드백, 실행 중인 코드를 통합 입력으로 받아 두 언어의 코드를 동시에 생성하는 파이프라인을 설계하였다. 둘째, 생성 코드의 컴파일 및 안전성 검증, 오류를 모델에 되돌려 다시 생성하게 하는 자동 수정 과정, 이전 코드 복원 등 공연 환경에 필요한 안전장치를 갖춘 시스템을 구현하였다. 셋째, 절대적, 상대적, 음악적 명령의 세 유형으로 구성된 벤치마크로 음성 입력 방식, AV 출력 제공 여부, 모델 선택이 생성 품질과 응답 시간에 미치는 영향을 정량 평가하였다.

본 논문의 구성은 다음과 같다. II장에서 관련 연구를 살펴보고, III장에서 제안 시스템의 설계를 기술한다. IV장에서 시스템 구현과 평가 결과를 제시하고, V장에서 결론을 맺는다.

II. 관련 연구

1. 라이브 코딩과 AV 퍼포먼스

라이브 코딩은 2000년대 초 실험적 전자음악 공연에서 출발해 하나의 예술 실천으로 발전하였다^[1]. 사운드 영역에

서 SuperCollider는 사운드를 합성하는 서버와 이를 제어하는 인터프리터 언어로 구성되어 있으며, 코드를 다시 평가하는 것만으로 연주 중인 사운드를 즉시 교체할 수 있어 라이브 코딩의 표준적 도구가 되었다^[3]. TidalCycles는 SuperCollider를 사운드 엔진으로 사용하는 패턴 언어로, 리듬 패턴을 짧은 표현식으로 기술하는 고수준 문법을 제공하여 리듬 중심의 라이브 코딩을 대중화하였다^[4]. Sonic Pi는 교육 목적으로 설계된 라이브 코딩 환경으로, 단순화된 문법과 내장 신시사이저와 샘플을 제공하여 프로그래밍 경험이 없는 학습자도 코드로 음악을 만들 수 있도록 진입장벽을 크게 낮췄다^[12].

영상 공연 분야에서도 라이브 코딩 환경이 발전해 왔다. Hydra는 웹 브라우저에서 동작하는 영상 라이브 코딩 환경으로, 아날로그 비디오 신시사이저의 신호 흐름을 본뜬 함수 연쇄를 짧은 코드로 작성하면 이를 GLSL 셰이더로 변환하여 실시간 영상을 생성한다^[5]. Gibber는 사운드와 영상을 하나의 브라우저 환경에서 단일 언어(JavaScript)로 함께 다루는 라이브 코딩 환경으로, 사운드와 영상 코드를 한 화면에서 작성하고 서로 연동할 수 있게 하였다^[13]. 그러나 SuperCollider, TidalCycles, Sonic Pi, Hydra, Gibber의 다섯 환경 모두에서 연주는 여전히 코드 타이핑에 의존한다.

또한 사운드와 영상을 함께 지원하는 Gibber를 제외하면 각 환경은 사운드 또는 영상 한 도메인에 특화되어 있어, AV 퍼포먼스를 위해서는 두 개 이상의 언어와 환경을 병행해야 하는 부담이 남는다.

2. 생성형 AI 기반 코드 생성과 창작 도구

생성형 AI의 발전에 따라 자연어 프롬프트로부터 실용 수준의 코드 생성이 가능해지고 있다^[7]. 이러한 생성형 AI를 창작 도구와 결합하기 위한 연구가 활발히 진행되고 있다. 영상 분야에서 AI Co-Artist는 GPT-4를 활용하여 사용자와의 상호작용 속에서 GLSL 셰이더 애니메이션을 반복적으로 생성하는 프레임워크이며, 실시간 시각 피드백과 오디오 반응성을 결합한 공연을 가능하게 하였다^[8]. 하지만 입력이 텍스트와 GUI에 한정되어 있으며 사운드 코드를 생성하지 않는 한계가 있다. 또한 증강현실 환경에서 사용자가 바라보는 장면의 시각적 인상을 자연어 지시만으로 변형하기 위해 ShadAR가 제안되었다^[9]. ShadAR는 LLM으로 증강현실 화면에 적용할 셰이더를 생성하여 현실 장면의 색과 질감 같은 시각적 지각을 실시간으로 바꾸지만, 입력이 텍스트 지시에 머무르고 사운드를 다루지 않는다는 점에서 AI Co-Artist와 같은 한계를 공유한다. 사운드 분야에서는 기타 악보 데이터로 라이브 코딩용 반복 악구를 생성하는 LooperGP^[14]와 같이 기호 음악 생성을 라이브 코딩에 접목한 연구가 있다. 또한 Sonic Pi 코드 생성에서 코드 표현과 실제 소리 사이의 의미 간극이 크다는 분석^[10]은 텍스트만으로 음악적 의도를 전달하는 접근의 한계를 보여준다. 더 나아가 생성된 결과의 오류를 모델 스스로 정정하기 위한 연구도 제안되었다. Self-Refine은 모델이 생성한 출력에 대해 같은 모델이 스스로 피드백을 작성하고, 그 피드백을 반영하여 출력을 다시 생성하는 과정을 반복함으로써 별도의 추가 학습 없이 출력 품질을 개선하는 방법이다^[15]. Self-Debugging은 생성된 코드를 실제로 실행한 뒤 오류 메시지와 실행 결과를 모델에 되돌려 주어 모델이 자신의 코드를 스스로 진단하고 수정하게 하는 방법으로, 코드 생성의 정확도를 크게 높일 수 있음을 보였다^[16]. 이처럼 생성-실행-피드백을 반복하는 구조는 본 시스템의 오류 자동 수정 설계의 근거가 된다.

이상 기존 연구를 분석해 보면 다음과 같은 방향으로 AV 시스템의 개선이 필요하다. 첫째, 발화뿐 아니라 허밍과 박수 같은 비언어적 음악 요소까지 포함하는 음성을 텍스트 변환 없이 소리 그대로 입력하는 인터페이스가 필요하다. 둘째, 현재 AV 출력을 모델 입력으로 다시 반영하여 상대적 명령을 처리하게 할 필요가 있다. 셋째, 사운드와 영상 코드를 한 번의 생성에서 함께 만들어 두 도메인의 음악적 일관성을 생성 시점부터 연계할 필요가 있다. 이 세 가지 요소를 통합하여 AV 시스템을 개선한다면, 생성형 AI를 활용하여 라이브 코딩 퍼포먼스를 보다 쉽고 직관적으로 연출하는 것을 기대할 수 있다. 다음 장에서 차별화된 요소를 바탕으로 한 생성형 AI 기반 AV 퍼포먼스 시스템을 소개한다.

III. 시스템 설계

기존의 AV 퍼포먼스를 개선하기 위해 생성형 AI 기반 라이브 코딩 퍼포먼스 시스템을 제안한다. II장에서 도출한 개선 방향에 따라 제안하는 시스템은 세 가지 요구사항을 중심으로 설계되었다. 첫째, 연주자가 공연 중 타이핑에서 벗어날 수 있도록, 발화뿐 아니라 허밍과 박수 같은 비언어적 입력까지 텍스트 변환 없이 처리하는 음성 인터페이스를 지원한다. 둘째, “지금보다 화면을 어둡게 해 줘”와 같이 현재 AV 출력을 기준으로 삼는 상대적 명령을 해석할 수 있도록, 시스템이 생성하고 있는 현재의 출력(화면과 사운드)을 다시 생성형 AI의 입력으로 되돌리는 AV 출력 피드백 구조를 포함한다. 셋째, 사운드(SuperCollider)와 영상(GLSL) 코드를 한 번의 생성으로 함께 만들어 두 도메인의 일관성을 생성 시점부터 보장하며, 생성된 코드가 공연을 중단시키지 않도록 코드 검증과 코드 복원 같은 안전장치를 갖춘다.

그림 2는 제안하는 생성형 AI 기반 AV 퍼포먼스 시스템의 전체 구조를 나타낸다. 시스템은 ①연주자의 음성 명령, AV 출력 피드백, 실행 중인 코드 파일을 수집하는 멀티모달 입력 단계, ②수집된 입력을 시스템 프롬프트로 설계된 생성형 AI에 전달하여 사운드(SuperCollider)와 영상(GLSL) 코드를 동시에 생성하는 AV 생성 단계, ③생성된

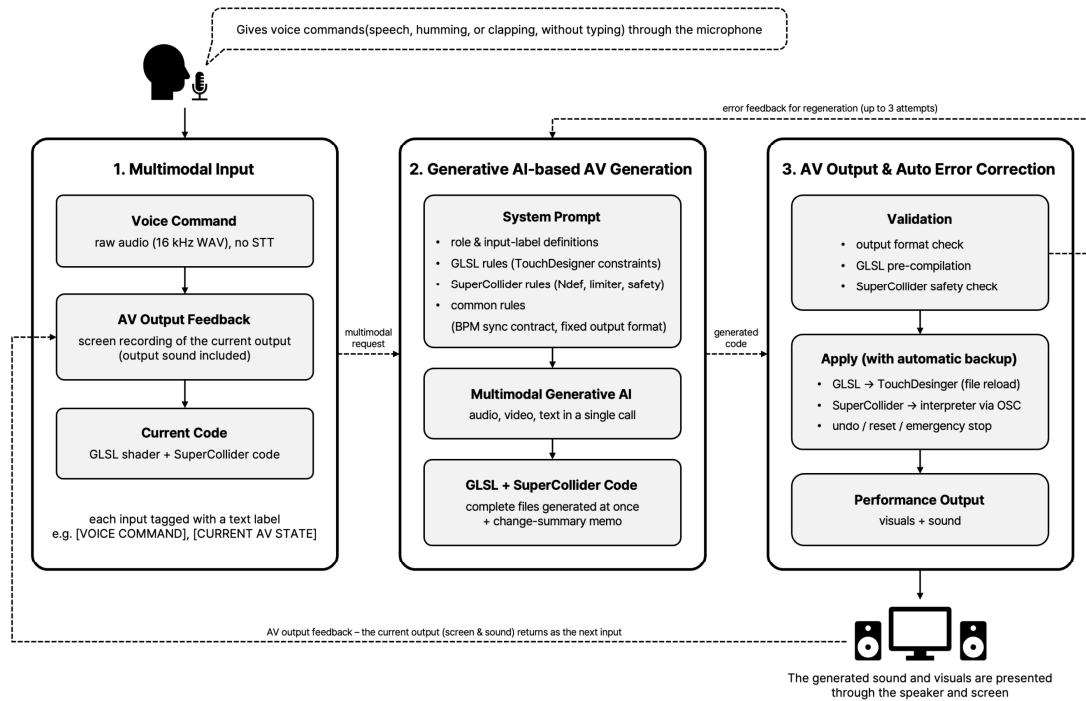


그림 2. 생성형 AI 기반 오디오비주얼 퍼포먼스 시스템의 전체 구조

Fig. 2. Overall architecture of the proposed AV performance system

코드를 검증하여 공연에 반영하고 오류 발생 시 자동으로 수정하는 AV 출력 단계의 세 부분으로 구성되며, 전체 과정은 독립된 제어 서버(Python, FastAPI)가 담당한다. 공연의 출력은 출력 단계에서 끝나지 않고 다음 명령의 입력으로 되돌아간다. 즉 시스템의 출력(화면과 사운드)이 AV 출력 피드백으로서 다시 멀티모달 입력 단계에 포함되는 순환 구조를 이루며, 이 구조가 현재 AV 출력을 기준으로 삼는 상대적 명령을 해석할 수 있게 한다. 영상은 노드 기반 실시간 그래픽 제작 도구인 TouchDesigner^[17]가 GLSL 셰이더를 실행해 렌더링하고, 사운드는 SuperCollider가 합성한다. 제어 서버와 SuperCollider는 음악 소프트웨어 간 통신에 널리 쓰이는 네트워크 프로토콜인 OSC(Open Sound Control)로 연결된다. 생성형 AI 호출처럼 수 초가 걸리는 작업은 렌더링과 분리된 별도 프로세스에서 처리되므로, 생성이 진행되는 동안에도 공연 중인 영상과 사운드는 끊기지 않는다. 연주자가 명령을 마치면 시스템은 곧바로 멀티모달 요청을 조립해 생성형 AI를 호출하고, 응답으로 받은 두 코드 파일을 검증한 뒤 즉시 공연에 반영한다.

1. 멀티모달 입력 구성

연주자가 명령을 내리면 시스템은 세 종류의 입력을 수집하여 하나의 요청으로 조립한다. ①음성 명령: 마이크로 녹음한 연주자의 발화(16 kHz 모노 WAV)를 텍스트로 변환하지 않고 소리 그대로 넣는다. ②AV 출력 피드백: 명령을 말하는 동안의 공연 출력을 담은 화면 녹화 영상으로, 화면과 함께 출력 사운드가 포함되어 있어 하나의 영상 입력만으로 지금 보이는 것과 들리는 것을 모두 전달한다. ③현재 코드: 실행 중인 GLSL 셰이더와 SuperCollider 코드 전문으로 구성된다.

각 입력 앞에는 그 역할을 알리는 텍스트 라벨(예: [VOICE COMMAND], [CURRENT AV STATE])을 붙여, 모델이 어떤 소리가 명령이고 어떤 소리가 출력 피드백인지 구분하게 한다. 생성 모델은 오디오와 비디오를 별도 전처리 없이 직접 입력받는 멀티모달 모델인 Gemini를 사용한다^[11]. 음성 인식(음성을 텍스트로 변환하는 단계)을 거치지 않는 이 단일 호출 구성은 처리 단계를 줄여 응답을 빠르

게 할 뿐 아니라, 발화에 섞인 허밍, 박수, 비트박스 같은 비언어적 음악 요소를 손실 없이 모델에 전달한다는 점에서 매우 중요하게 작용한다.

2. 시스템 프롬프트 설계 및 생성형 AI 기반 AV 생성

본 시스템의 생성형 AI는 범용 모델을 그대로 호출하는 것이 아니라, 라이브 코딩 퍼포먼스 보조라는 목적에 맞게 시스템 프롬프트로 동작 규칙을 미리 설계해 둔 상태로 동작한다. 시스템 프롬프트는 연주자의 명령마다 새로 작성되는 것이 아니라 시스템 설계 단계에서 고정된 것으로, 모델이 어떤 입력을 받아 어떤 형식의 코드를 생성해야 하는지를 규정하는 본 시스템의 핵심 설계 요소다. 시스템 프롬프트는 네 부분으로 구성하였다. 첫째, 역할과 상황은 무대 위 라이브 코더를 보조하는 코드 생성 엔진이라는 역할, 그리고 각 입력 라벨의 의미를 정의한다. 둘째, GLSL 규칙은 TouchDesigner의 셰이더 실행 환경에 맞는 제약(버전 선언 금지, 미리 연결된 변수만 사용, 지정된 출력 함수 사용 등)과, 모든 셰이더가 포함해야 하는 고정 선언부를 강제한다. 셋째, SuperCollider 규칙은 사운드 서버의 시작과 종료 금지, 코드를 다시 평가할 때 소리가 겹겹이 쌓이지 않고 교체되도록 이름 붙인 사운드 정의(Ndef)를 사용할 것, 출력 음량 상한과 리미터 적용, 파일 삭제나 외부 명령 실행 같은 위험 동작 금지를 명시한다. 넷째, 공통 규칙은 명령이 요구한 부분만 고치고 나머지는 보존할 것(공연의 연속성 유지), 두 코드가 같은 템포(BPM) 값을 공유할 것(사운드-영상 동기화 계약), 그리고 응답을 미리 정한 구분 기호(<<<GLSL>>>, <<<SCD>>> 등)로 감싼 고정 형식으로 출력할 것을 규정한다.

생성 단계에서 모델은 조합된 멀티모달 입력을 바탕으로 명령의 의도를 해석하고, 그 의도를 현재 코드 위에 반영한 새 코드를 만들어 낸다. “120 BPM의 킥 드럼을 넣어 줘”처럼 현재 AV 출력과 무관한 절대적 명령은 발화 내용만으로 해석되지만, “지금보다 화면을 어둡게 해 줘”처럼 현재 AV 출력을 기준으로 삼는 상대적 명령은 AV 출력 피드백에 담긴 화면 밝기나 템포를 기준으로 삼아 변화량이 결정된다. 나아가 허밍에 이어 “킥을 이 리듬으로 바꿔 줘”라고 지시하는 것과 같은 음악적 명령에서는, 모델이 음성에 담긴 허밍

이나 박수의 리듬과 음높이를 직접 해석하여 킥 패턴이나 베이스 멜로디 같은 사운드 코드의 음악적 내용으로 변환한다. 이는 음성을 텍스트로 변환하지 않고 소리 그대로 입력하는 설계가 생성 단계에서 실질적 효과를 내는 지점이다.

이 설계에 따라 생성형 AI는 사운드(SuperCollider)와 영상(GLSL) 코드를 한 번의 호출에서 동시에 생성한다. 두 코드를 별개의 요청으로 만들지 않고 함께 생성하는 것은, 시스템 프롬프트의 동기화 계약과 결합하여 두 도메인이 같은 템포와 음악적 의도를 공유하도록 생성 시점부터 보장하기 위한 설계다. 모델은 매번 두 파일의 완전한 새 버전을 출력한다. 바뀐 부분만 출력하게 하는 방식보다 응답이 길어지지만, 부분 수정 지시가 잘못 적용되어 코드가 깨지는 위험이 없어 공연 상황에 더 적합하다고 판단하였다. 응답은 구분 기호를 기준으로 두 코드와 변경 요지 메모로 분리되어 다음 절의 출력 단계로 넘어간다.

3. AV 출력과 자동 오류 수정

생성된 코드는 적용 전에 네 단계의 검증을 거친다. ①형식 검사 단계는 응답이 규정된 출력 형식을 지켰는지 확인한다. ②GLSL 컴파일 검사 단계는 화면 출력 없이 GPU 컴파일만 수행하는 별도 환경에서 셰이더를 미리 컴파일해, 문법 오류를 공연 화면에 닿기 전에 걸러낸다. ③SuperCollider 안전성 검사 단계는 파일 삭제, 외부 명령 실행, 사운드 서버 종료처럼 공연을 중단시키거나 시스템을 훼손할 수 있는 코드를 금지 목록과 대조해 차단한다. ④실제 적용 단계는 검사를 통과한 코드를 각 실행 환경에 적용하고, 이때 발생하는 실행 오류까지 수집한다. 새 GLSL 코드는 파일로 저장되며, TouchDesigner가 파일 변경을 감지해 자동으로 셰이더를 다시 컴파일하므로 화면이 즉시 갱신된다. 새 SuperCollider 코드는 OSC를 통해 인터프리터로 전달되어 평가되고, 연주 중인 사운드를 교체한다.

어느 단계에서든 오류가 나오면 시스템은 오류 내용을 모델에 그대로 되돌려 주고 수정본을 다시 생성하게 한다. 이전 응답을 대화 맥락에 유지한 채 오류만 전달하므로 모델은 직전 출력을 최소한으로 고치는 방향으로 수정한다. 이 과정은 최초 생성을 포함해 최대 3회까지 자동 반복되며, 모델 스스로 출력을 반복 개선하게 하는 기존 연구^{[15][16]}의

접근을 공연 파이프라인에 맞게 적용한 것이다. 자동 반복으로 해소되지 않은 오류는 콘솔의 수정 버튼(FIX)으로 연주자가 원하는 시점에 같은 과정을 다시 시작할 수 있다. 모든 적용 직전에는 현재 코드가 자동으로 백업되어, 연주자는 버튼 하나로 직전 코드로 되돌리거나(실행 취소) 공연 시작 시점의 초기 코드로 복원할 수 있다. 또한 생성이 진행 중일 때도 동작하는 긴급 정지 기능을 별도 통신 경로로 두어, 의도치 않은 소리가 났을 때 모든 사운드를 즉시 멈출 수 있게 하였다.

출력 단계에서는 생성 시점의 동기화 계약을 보완하는 실행 시점의 사운드-영상 동기화가 함께 이루어진다. SuperCollider의 출력 사운드를 가상 오디오 장치를 통해 TouchDesigner로 전달하고, TouchDesigner가 이 소리를 실시간 분석한 7개 값(전체 음량, 저·중·고 대역별 세기, 킥과 스네어

와 리듬의 타격 신호)을 매 화면 프레임마다 셰이더의 입력 변수로 넣어준다. 모든 생성 셰이더는 시스템 프롬프트가 강제하는 고정 선언부를 통해 이 값들을 즉시 사용할 수 있고, 오디오 분석이 연결되지 않은 환경에서는 시간 기반 값으로 대신 동작하도록 규칙을 두었다. 실제 소리를 분석해 영상을 구동하는 이 방식은, 생성 시점에 템포 상수를 공유하는 것만으로는 대응할 수 없는 미세한 박자 어긋남에도 사운드와 영상이 함께 움직이도록 한다.

IV. 구현 및 평가

1. 시스템 구현

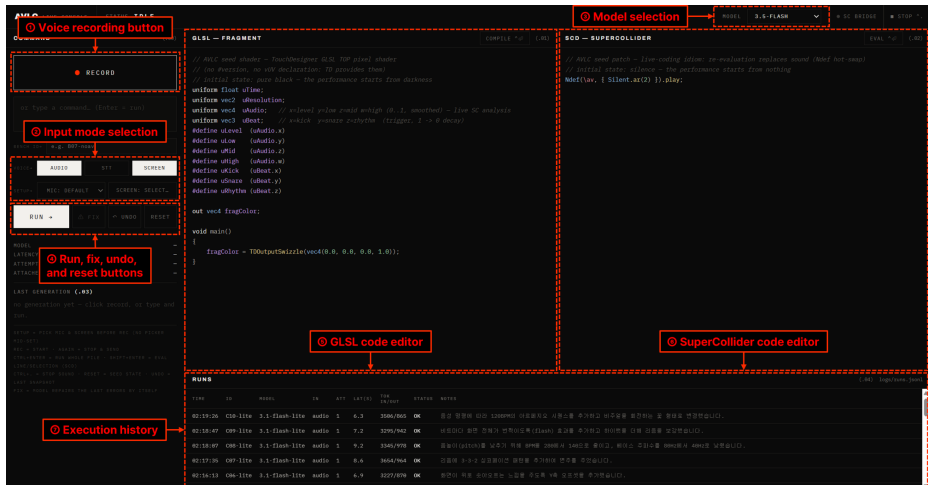


그림 3. 라이브 코딩 공연용 콘솔 화면
Fig. 3. Console interface for live coding performance

표 1. 구현 환경
Table 1. Implementation environment

Components		Specification
Multimodal input	Voice input	16 kHz mono WAV, 16-bit PCM
	Transcription model	gemini-3.1-flash-lite, temperature 0.0
Generative AI	Generation model	gemini-3.5-flash (default), gemini-3.1-flash-lite
	Generation settings	Temperature 0.4, max output 8,192 tokens, thinking disabled
AV rendering	Visual renderer	TouchDesigner 2025.32280, GLSL TOP
	Sound engine	SuperCollider 3.14.1 (SCSynth + SCLang, OSC)
Automatic error correction		Up to 3 attempts (1 initial + 2 corrections)
AV output feedback input		Screen recording with sound (WebM, 720p, 15 fps)
Control server		Python 3.11.9, FastAPI
Hardware		Intel Core i9-14900HX, NVIDIA GeForce RTX 4080 Laptop GPU, 64 GB RAM

제안하는 생성형 AI 기반 AV 퍼포먼스 시스템은 III장의 설계에 따라 멀티모달 입력, 멀티모달 생성형 AI(Gemini API), 영상 렌더러인 TouchDesigner, 사운드 서버인 SuperCollider, 그리고 전체 파이프라인을 담당하는 파이썬(FastAPI) 제어 서버로 구현되었다. 세부 구현 환경은 표 1과 같다.

위의 환경을 바탕으로 구현된 시스템은 브라우저에서 동작하며 공연자는 공연용 콘솔 인터페이스를 통해 사용한다. 그림 3은 공연용 콘솔의 화면 구성을 나타낸다. ①음성 녹음 버튼은 버튼 한 번으로 녹음을 시작하고 종료하며, 녹음 동안 공연 화면과 출력 사운드가 함께 기록되어 AV 출력 피드백으로 사용된다. ②입력 방식 선택은 음성 원본 입력과 텍스트 변환(STT) 입력, 화면 녹화 포함 여부를 전환하는 부분이다. ③모델 선택은 생성에 사용할 모델을 전환하

는 메뉴이다. ④실행 및 수정 버튼은 조립된 명령을 실행하는 RUN, 남은 오류의 수정을 다시 시작하는 FIX, 직전 코드 복원과 초기 코드 복원을 수행하는 UNDO와 RESET 버튼을 제공한다. ⑤GLSL 코드 편집기와 ⑥SuperCollider 코드 편집기는 생성형 AI가 생성한 코드가 적용될 때마다 이 편집기에 반영되어, 연주자는 현재 공연을 구동하는 두 코드를 문법 색상 표시와 함께 실시간으로 확인하고 직접 수정할 수도 있다. ⑦실행 기록은 시행별 사용 모델, 입력 조건, 시도 횟수, 응답 시간, 토큰 사용량과 변경 요지를 시간 순으로 보여준다.

그림 4는 구현된 시스템의 AV 생성 과정과 결과를 나타내고 있다. 초기 상태(검은 화면과 무음)에서 연주자가 “몽환적인 패드 사운드와 킥 드럼을 넣고, 노이즈가 일렁이는 배경 위에 기하학적인 패턴이 움직이는 비주얼을 만들어

```

GLSL - FRAGMENT
#define uRhythm (uBeat.2)

out vec4 fragColor;

// 2D Noise helper functions for the undulating background
float hash(vec2 p) {
    return fract(sin(dot(p, vec2(127.1, 311.7))) * 43758.5453123);
}

float noise(vec2 p) {
    vec2 i = floor(p);
    vec2 f = fract(p);
    vec2 u = f * f * (3.0 - 2.0 * f);
    return mix(mix(hash(i + vec2(0.0, 0.0)), hash(i + vec2(1.0, 0.0)), u.x),
        mix(hash(i + vec2(0.0, 1.0)), hash(i + vec2(1.0, 1.0)), u.x), u.y);
}

float fbm(vec2 p) {
    float v = 0.0;
    float a = 0.5;
    vec2 shift = vec2(100.0);
    mat2 rot = mat2(cos(0.5), sin(0.5), -sin(0.5), cos(0.5));
    for (int i = 0; i < 4; ++i) {
        v += a * noise(p);
        p = rot * p * 2.0 + shift;
        a *= 0.5;
    }
    return v;
}

void main()
{
}
    
```

(a) Generated GLSL code

```

SCD - SUPERCOLLIDER
Ndef{lav, {
    var bpm = 115;
    var beat = Impulse.kr(bpm / 60);
    var t = PulseCount.kr(beat);

    // --- 1. KICK DRUM ---
    var kickTrig = Trig.kr(beat * (t % 2), 0.01);
    var kickEnv = EnvGen.ar(Env.perc(0.005, 0.4, curve: -0), kickTrig);
    var kickFreq = EnvGen.ar(Env([120, 45, 30], [0.015, 0.15], \exp), kickTrig);
    var kick = SinOsc.ar(kickFreq) * kickEnv * 1.2;

    // --- 2. DREAMY PAD ---
    // Multi-channel expansion inside the Ndef function using a fixed array
    var chord = [57, 60, 64, 67, 71]; // A#m chord
    var pad = Mix.new(chord.collect{|midi, i|
        var detune = LFNoise1.kr(0.2).bipolar(0.25);
        var osc1 = LFTri.ar((midi + detune).midicps);
        var osc2 = LFSaw.ar((midi - detune + 0.05).midicps);
        var lfo = SinOsc.kr(0.1 * (i * 0.05)).range(0.2, 0.8);
        (osc1 * lfo) + (osc2 * (1.0 - lfo))
    }) * 0.04;

    var padEnv = LFNoise1.kr(0.3).range(0.4, 0.9);
    var filterCutoff = SinOsc.kr(0.001).expRange(400, 2500);

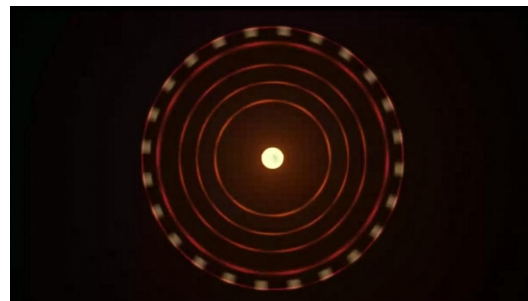
    pad = pad * padEnv;
    pad = LPF.ar(pad, filterCutoff, 0.8);

    // Stereo widening and reverb
    pad = Pan2.ar(pad, LFNoise1.kr(0.1).range(-0.3, 0.3));
    pad = FreeVerb.ar(pad, mix: 0.5, room: 0.05, damp: 0.3);
}
}
    
```

(b) Generated SuperCollider code



(c) Final output screen (raw audio)



(d) Final output screen (raw audio + AV output feedback)

그림 4. 음성 명령에 대한 생성 결과 예시

Fig. 4. Example of generation results for a voice command

줘”라고 음성으로 명령을 입력하였다. 모델은 Super-Collider 코드에 몽환적인 패드와 킥 드럼의 사운드 정의(Ndef)를 생성하고(그림 4 (b)), GLSL 코드에 노이즈 기반의 일렁이는 배경과 그 위에서 움직이는 기하학적 패턴을 생성하되(그림 4 (a)), 시스템 프롬프트의 고정 선언부를 통해 킥의 타격 신호가 셰이더 입력 변수로 연결되어 영상이 사운드에 반응하도록 만든다. 생성된 두 코드는 검증 과정을 거쳐 즉시 적용되는데 그림 4 (c)는 음성 명령을, 그림 4 (d)는 음성 명령과 AV 출력 피드백을 반영하여 생성 및 출력된 AV 라이브 공연 화면을 나타내고 있다.

2. 실험 설계

제안하는 생성형 AI 기반 라이브 코딩 퍼포먼스 시스템의 성능을 파악하기 위해 세 가지 실험 조건을 구성하였다. 첫 번째 실험 조건(E1)은 생성 모델에 따른 특성을 파악하기 위해 기준 모델(gemini-3.5-flash)과 경량 모델(gemini-3.1-flash-lite)을 각각 적용하여 공연 결과가 생성되도록 구성하였다. 두 번째 실험 조건(E2)은 첫 번째 결과에서 나타난 적절한 모델을 바탕으로 입력 방식에 따른 특성을 파악하기 위해 음성 명령을 직접 입력으로 활용하는 방법과 일반적인 음성 인터페이스처럼^[8] 텍스트로 변환한 뒤 입력하는 방식으로 공연 결과가 생성되도록 하였다. 세 번째 실험 조건(E3)은 시스템의 AV 출력 피드백이 최종 AV 공연 영상 생성에 주는 영향을 분석하기 위해 AV 출력 피드백을 통제하였다.

세 가지 실험 시나리오에 대한 성능을 평가하기 위해 절대적 명령, 상대적 명령 및 음악적 명령 세 가지 유형의 공연 명령을 구성하였다. 절대적 명령(A형)은 “120 BPM의 킥 드럼을 넣어 줘”처럼 현재 AV 출력과 무관하게 해석할 수 있는 통제군이다. 상대적 명령(B형)은 “지금보다 화면을 어둡게 해 줘”처럼 현재 AV 출력을 기준으로만 해석되는 명령으로 E3를 변별한다. 음악적 명령(C형)은 리듬 허밍에 이어 “킥을 이 리듬으로 바꿔 줘”라고 지시하는 것처럼 말로 옮길 수 없는 음악 요소를 포함하는 명령으로 E2를 변별한다. 각 유형에 대해 10개씩 총 30개의 명령은 하나의 공연이 전개되듯 이어지도록 구성하였다. 시스템의 초기 상태는 검은 화면과 무음이며, 절대 명령들이 무에서 사운드

와 영상을 단계적으로 쌓아 올리고(킥 드럼 → 킥에 반응하는 시각 요소 → 드론 → 하이햇 ...), 이어지는 상대 명령과 음악적 명령이 그 위에서 동작한다. 이 구성은 상대 명령이 참조할 출력 피드백을 자연스럽게 보장할 뿐 아니라, 실제 라이브 코딩 공연이 빈 화면과 무음에서 시작해 점진적으로 장면을 구축하는 흐름을 그대로 모사한다.

성능 평가를 위해 네 가지 지표를 측정하였다. ①성공률: 생성 코드가 컴파일과 안전성 검사, 실행 평가를 모두 통과해 적용된 비율로, 자동 수정 이전(1차 시도)과 이후(최대 3회)를 구분해 오류 자동 수정의 기여를 함께 측정한다. ②의도 반영률: 적용된 결과가 명령의 의도를 실제로 반영했는지의 비율로, 시행마다 출력 화면과 사운드를 명령별 판정 기준(화면 평균 색과 밝기 변화, 출력 사운드의 템포와 스펙트럼 변화 등 관찰 가능한 신호)에 따라 저자가 이진 판정하며, 판정 근거가 되는 화면과 사운드 기록은 모두 보존하였다. ③응답 시간: 명령 입력이 끝난 시점부터 생성 코드가 공연에 반영될 때까지 한 번의 생성 시도가 걸린 시간으로, 자동 수정이 개입한 시행은 최종 시도를 기준으로 집계한다. ④토큰 사용량: 시행당 입력과 출력 토큰 수(재시도 포함 합계)로, 운용 비용의 근거가 된다.

평가는 하나의 조건을 하나의 세션으로 하여 진행한다. 각 조건에서 시스템을 초기 상태(검은 화면과 무음)로 되돌린 뒤 30개 명령을 고정된 순서(A01 → C10)로 연속 수행한다. 시행이 실패한 경우에는 직전 코드로 복원한 뒤 다음 명령을 진행하여 세션의 연속성을 유지한다. 명령 음성은 명령마다 한 번씩 사전에 녹음해 두고 모든 조건에서 같은 녹음을 재생하여 입력함으로써, 조건 간 발화 차이(특히 허밍의 리듬과 음높이 변동)에 따른 교란을 제거한다. 의도 반영률의 판정 기준은 실험 전에 명령별로 확정하여 네 조건에 동일하게 적용하였고, 판정은 조건 간 결과를 대조하지 않은 상태에서 조건별로 개별 수행하였다. 기준 조건(음성 원본 입력 + AV 출력 피드백 제공 + gemini-3.5-flash)의 세션을 먼저 수행한 뒤, 생성 모델만 경량 모델로 교체한 조건(E1), 같은 음성을 텍스트로 변환해 입력하는 조건(E2), 출력 피드백 입력만 제거한 조건(E3)을 각각 별도 세션으로 재수행한다. 이에 따라 전체 평가는 기준 조건과 세 개의 비교 조건, 총 4개 조건 120회 시행으로 구성되며, 자동 수정 재시도를 포함한 실제 생성 호출은 140회였다.

표 2. 조건별 성능 비교 결과

Table 2. Performance comparison across all conditions

Metric	gemini-3.1-flash-lite	gemini-3.5-flash		
	Baseline model (E1) (raw audio + output feedback)	Lightweight model (E1) (raw audio + output feedback)	Transcribed text (E2)	Without output feedback (E3)
Success rate, 1st attempt (%)	93.3	86.7	90.0	83.3
Success rate, after auto-correction (%)	100.0	100.0	100.0	93.3
Intent-reflection rate, overall (%)	66.7	93.3	63.3	63.3
Intent-reflection, type A / B / C (%)	80.0 / 80.0 / 40.0	90.0 / 100.0 / 90.0	90.0 / 90.0 / 10.0	60.0 / 70.0 / 60.0
Response time, mean $\pm$ SD (s)	8.4 $\pm$ 3.9	14.2 $\pm$ 5.6	12.7 $\pm$ 6.0	11.8 $\pm$ 5.7
Output tokens per run (mean)	947	2,680	2,781	2,604

3. 평가 결과

평가 결과를 분석하기 위해 성공률과 응답 시간, 토큰 사용량은 실행 로그에서 산출하였고, 의도 반영률은 시행마다 출력을 보고 들으며 판정하여 집계하였다. 적용에 실패한 시행과, 적용은 되었으나 출력 사운드가 손상되어 사용할 수 없었던 시행의 의도 반영은 모두 미반영으로 집계하였다. 표 2는 네 조건의 결과를 함께 나타내고 있다. 기준 구성(Baseline)은 음성 원본 입력과 AV 출력 피드백을 모두 제공한 조건이며, 이하에서 생성 모델(E1), 음성 입력 방식(E2), 출력 피드백 제공 여부(E3)의 영향을 차례로 분석하고 오류 자동 수정의 기여를 살펴본다.

3.1 생성형 AI 모델 비교 (E1)

동일한 AV 생성에서 생성 모델을 경량 모델로 적용하면 평균 응답 시간이 8.4초로 기준 모델보다 1.7배 빠르고 호출당 출력 토큰이 1/3 수준으로 운용 비용 면에서 유리한 것으로 나타났다. 1차 시도 성공률도 93.3%로 기준 모델(86.7%)보다 높았다. 실제로 경량 모델이 생성한 코드는 GLSL 38.5줄, SuperCollider 13.0줄로 기준 모델(각각 91.5줄, 39.1줄)의 1/2에서 1/3 수준이었고, 시행 간 표준편차(4.2와 3.1)도 기준 모델(19.8과 13.3)보다 작아 명령과 무관하게 짧고 정형화된 코드를 반복 생성하였다. 다만 이 성공률 차이는 30회 시행에서 1차 실패가 각각 2건(6.7%)과 4건(13.3%)으로 두 건 차이에 해당하므로, 모델 간 코드 생성 능력의 우열로 해석하기는 어렵다. 그러나 명령의 의도를 실제로 반영했는지를 보면 전체 의도 반영률은 66.7%에

그쳤고, 허밍과 박수를 해석해야 하는 음악적 명령(C형)에서는 40%까지 떨어졌다. 오류가 없고 의도가 대체로 적합한 시행에서도 결과물의 표현 밀도가 낮았는데, 겹겹의 레이어가 쌓인 기준 모델의 장면과 달리 단일 도형 수준의 화면이 만들어졌다. 이는 앞서 확인한 코드 분량 차이와도 부합한다. 경량 모델은 빠른 스케치용으로는 유용하지만, 공연의 표현 밀도를 유지하려면 상위 모델이 우수함을 알 수 있다.

3.2 음성 입력 방식 비교 (E2)

같은 음성 명령을 텍스트로 변환한 뒤 텍스트만 입력한 조건에서, 성공률은 기준 구성과 유사했으나 전체 의도 반영률이 93.3%에서 63.3%로 떨어졌고 하락은 음악적 명령(C형)에 집중되었다(90% $\rightarrow$ 10%). 이러한 원인은 텍스트 변환에 따른 정보 손실과 전사 인터페이스의 정확도 두 가지로 나뉜다. 먼저 정보 손실에 해당하는 경우는 허밍과 박수가 “[humming]”, “[claps]” 같은 표기로만 남아 리듬과 음높이 정보가 완전히 사라졌다(C01, C02, C04, C07, C10). 전사 정확도에 해당하는 경우는 허밍이 아예 언급조차 남지 않거나(C03: “베이스가 이 멜로디를 따라가게 해줘”라는 문장만 남아 멜로디 정보가 전무), 템포 허밍이 “템포를 2 빠르게 맞춰줘”로 오전사(C05)되며 발생하였다. 발화 명령에서도 전사 오류가 그대로 전파되어, “번쩍임을...”이 “잔향임을...”로 변환되면서 영상 명령이 사운드 수정으로 실행된 사례(B09)가 나타났다. 반면 음성 원본 조건에서 모델은 허밍의 리듬과 음높이를 직접 해석해 킥 패턴, 베이스 멜로디, 드론 음높이에 반영했다. 토큰 사용량은 두 조건

사이에 거의 차이가 없어, 음성 원본 입력에 추가적인 컴퓨팅 부담이 발생하지 않았다. 관찰된 사례의 다수는 정보 손실에 해당하며, 이는 전사 정확도를 높이더라도 해소되지 않는다. 허밍의 리듬과 음높이를 문자로 옮기려면 음악 기보에 준하는 표현 체계가 필요한데, 이는 일반적인 음성-텍스트 변환 인터페이스의 출력 형식이 아니기 때문이다. 앞선 명령의 맥락과 결합하면 “[humming]”의 대략적인 성격은 추정할 수 있으나, 코드 생성에 필요한 것은 킥 패턴의 박 위치나 베이스의 음정 같은 구체적 값이므로 맥락만으로는 복원되지 않는다.

3.3 출력 피드백 제공 여부 비교 (E3)

AV 출력 피드백을 제거한 조건에서도 전체 의도 반영률은 63.3%로 떨어졌으며 명령별 평가에서 상대 명령(B형)의 하락이 뚜렷했다(100% → 70%). 이러한 이유는 현재 템포를 듣지 못한 채 “지금 템포를 두 배로”를 실행하게 되면 하이햇이 분간되지 않을 만큼 빨라지거나(B02), “번쩍임을 지금보다 은은하게”가 화면을 거의 꺼버리는 수준의 어두움으로 실행되고(B09), “지금보다 소리를 부드럽게”가 오히려 더 거친 결과로(B08) 확인되었다. 이는 기준점을 지각하지 못한 모델이 변화량을 추측하기 때문이다. 주목할 것은 하락이 상대 명령에 국한되지 않았다는 점인데, 절대 명령(90% → 60%)과 음악적 명령(90% → 60%)도 함께 떨어졌고, SuperCollider 평가 오류는 3건에서 10건으로 늘었으며, 자동 수정으로도 회복하지 못한 실패 2건이 이 조건에서만 발생해 수정 후 성공률도 유일하게 100%에 도달하지 못했다(93.3%). 즉 AV 출력 피드백은 상대 명령의 기준점 일 뿐 아니라, 수정이 기존 장면과 일관되도록 붙잡아 주는 맥락으로도 기능함을 의미한다. 실제로 이 조건의 최종 화면은 세션이 쌓아 온 동심원 장면에서 이탈한 격자 패턴으로 표류했다. 한편, 입력이 작아진 만큼 응답은 11.8초로 다소 빨라졌으나, 토큰 사용량은 오류와 재시도가 늘면서 기준 구성과 거의 차이가 없었다. 결국 출력 피드백을 생략하는 것은 응답 시간을 개선할 수 있지만 대신 품질을 상대적으로 크게 떨어뜨리는 영향이 있음을 알 수 있다.

3.4 오류 분석과 자동 수정의 기여

추가적으로 검증 단계별 오류 발생 건수를 분석하였다.

GLSL 사전 컴파일 실패는 전혀 발생하지 않았는데, 이는 시스템 프롬프트가 강제하는 고정 선언부와 작성 규칙이 셰이더 생성을 안정화했음을 의미한다. 반면 오류의 대부분(21/22건)은 SuperCollider 평가 단계에서 발생하여, 실제 인터프리터 평가를 검증 경로에 포함한 설계가 유효했음을 확인하였다. 오류가 발생한 14회의 시행 중 12회가 자동 수정으로 재적용에 도달하여 최종 성공률은 전체 118/120회(98.3%)였다. 다만 재적용된 시행 중 3회는 출력 사운드가 손상되어 공연에 사용할 수 없었고 1건은 적용 직후 사운드 서버가 정지하여, 적용 시점의 검증만으로는 음향적 품질까지 보장하지 못한다는 한계도 확인하였다. 또한 재시도가 개입하면 대기 시간이 누적되어, 기준 구성의 평균 응답 시간은 최종 시도 기준 14.2초에서 재시도를 포함하면 15.9초로 늘어났다.

V. 결 론

본 논문은 라이브 코딩 공연을 위해 연주자의 음성 명령, 현재 공연의 AV 출력, 실행 중인 소스 코드를 멀티모달 생성형 AI에 한 번에 입력하여 사운드(SuperCollider)와 영상(GLSL) 코드를 동시에 생성하고 적용하는 AV 라이브 코딩 시스템을 설계하고 구현하였다. 제한한 시스템은 음성을 텍스트로 변환하지 않고 소리 그대로 입력하는 설계로 허밍과 박수 같은 비언어적 음악 요소를 명령으로 쓸 수 있게 하였고, 현재 AV 출력을 모델에 제공하여 “지금보다” 형태의 상대적 명령을 해석할 수 있게 하였다. 또한 컴파일 및 안전성 검증, 오류 자동 수정, 이전 코드 복원과 긴급 정지를 갖춰 공연 환경이 요구하는 안정성을 확보하였다. 세 유형 30종의 벤치마크 실험 결과, 자동 수정을 포함한 파이프라인 성공률은 98.3%였고, 기준 구성의 의도 반영률 93.3%는 텍스트 변환 입력과 AV 출력 미제공 조건을 크게 앞섰다. 특히 허밍과 박수로 제시한 음악적 명령은 원본 오디오 입력에서만 유의미하게 반영되어, 음성 원본 입력과 현재 AV 출력 피드백이 유용함을 확인하였다.

본 연구의 한계는 시스템과 평가 방법 두 측면으로 나누어 볼 수 있다. 먼저 시스템 측면에서 보면 GLSL 사전 컴파일 검사는 실제 렌더링 환경을 근사한 것이어서 환경 오류

의 오류까지 모두 걸러내지 못하였고, SuperCollider 위험 코드 차단은 금지 목록 방식이어서 완전한 격리 실행이 아니므로 연주자의 감도를 전제로 한다. 또한 실행 평가를 통과해 적용된 코드가 실제로는 사운드에 손상되어 있거나 적용 직후 서버가 정지될 수 있어 현재의 적용 시점 검증만으로는 음향적 품질까지 보장할 수 없으며, 적용 후 출력 신호를 일정 시간 감시해 이상 시 자동 복원하는 사후 검증 단계가 필요하다. 마지막으로 생성형 AI의 비결정성 때문에 같은 명령으로 일관성 있는 결과를 얻기 힘들는데, 이는 공연 예술의 즉흥성 관점에서 표현 자원이 될 수도 있으나 예측 가능성 측면에서는 제약으로 볼 수 있다. 평가 방법 측면에서는 의도 반영률의 판정이 사전에 확정한 기준을 따랐으나 단일 평가자에 의해 이루어졌고, 모델 비교는 동일 계열의 두 모델에, 음성 입력 방식 비교는 단일 전사 인터페이스에 한정되었다. 또한 평가가 통제된 실내 환경에서 수행되어 공연장의 소음 환경은 반영되지 않았다. 실제 공연장은 확성 사운드와 관객 소음이 상존하므로 명령 음성의 명료도가 낮아질 뿐 아니라, 음성을 소리 그대로 입력하는 본 시스템에서는 배경음이 명령 구간에 섞여 명령과 출력 피드백의 경계가 흐려질 수 있다.

향후 과제로는 복수 평가자의 교차 판정과 멀티모달 AI 기반 자동 평가를 통한 의도 반영률 재검증, 모델군과 전사 인터페이스를 확장한 비교, 실제 라이브 코더를 대상으로 한 사용자 연구를 통한 정성 평가, 그리고 변경된 부분만 생성하는 방식의 응답 시간 단축을 수행할 예정이다. 나아가 지향성 마이크와 음향 반향 제거를 적용하되 허밍과 박수 같은 비언어적 음악 요소는 보존하는 잡음 대응을 갖추어 실제 공연 무대에서 검증하고자 한다.

참 고 문 헌 (References)

- [1] N. Collins, A. McLean, J. Rohrerhuber, and A. Ward, "Live coding in laptop performance," *Organised Sound*, Vol.8, No.3, pp.321-330, Dec. 2003.
doi: <https://doi.org/10.1017/S135577180300030X>
- [2] A. F. Blackwell and N. Collins, "The programming language as a musical instrument," *Proceedings of the 17th Workshop of the Psychology of Programming Interest Group (PPIG)*, Brighton, UK, pp.120-130, 2005, <https://ppig.org/papers/2005-ppig-17th-blackwell/> (accessed Aug. 31, 2026).
- [3] J. McCartney, "Rethinking the computer music language: SuperCollider," *Computer Music Journal*, Vol.26, No.4, pp.61-68, Dec. 2002.
doi: <https://doi.org/10.1162/014892602320991383>
- [4] A. McLean, "Making programming languages to dance to: live coding with Tidal," *Proceedings of the 2nd ACM SIGPLAN International Workshop on Functional Art, Music, Modeling & Design (FARM)*, Gothenburg, Sweden, pp.63-70, 2014.
doi: <https://doi.org/10.1145/2633638.2633647>
- [5] O. Jack, Hydra - live coding video synth, <https://hydra.ojack.xyz> (accessed Jul. 12, 2026).
- [6] N. N. Correia and A. Tanaka, "AVUI: Designing a toolkit for audiovisual interfaces," *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, Denver, CO, USA, pp.1093-1104, 2017.
doi: <https://doi.org/10.1145/3025453.3026042>
- [7] M. Chen et al., "Evaluating large language models trained on code," arXiv preprint arXiv:2107.03374, 2021.
doi: <https://doi.org/10.48550/arXiv.2107.03374>
- [8] K. A. Yuksel and H. Sawaf, "AI Co-Artist: A LLM-powered framework for interactive GLSL shader animation evolution," arXiv preprint arXiv:2512.08951, 2025.
doi: <https://doi.org/10.48550/arXiv.2512.08951>
- [9] Y. Mei, S. Wendt, J. Wombacher, F. Müller, and J. Gugenheimer, "ShadAR: LLM-driven shader generation to transform visual perception in augmented reality," *Extended Abstracts of the 2026 CHI Conference on Human Factors in Computing Systems*, Barcelona, Spain, pp.1-4, 2026.
doi: <https://doi.org/10.1145/3772363.3799378>
- [10] S. Kouteili, H. Madhu, G. Typaldos, and M. Santolucito, "Embedding alignment in code generation for audio," arXiv preprint arXiv:2508.05473, 2025.
doi: <https://doi.org/10.48550/arXiv.2508.05473>
- [11] Gemini Team, Google, "Gemini: A family of highly capable multimodal models," arXiv preprint arXiv:2312.11805, 2023.
doi: <https://doi.org/10.48550/arXiv.2312.11805>
- [12] S. Aaron and A. F. Blackwell, "From Sonic Pi to Overtone: Creative musical experiences with domain-specific and functional languages," *Proceedings of the 1st ACM SIGPLAN Workshop on Functional Art, Music, Modeling & Design (FARM)*, Boston, MA, USA, pp.35-46, 2013.
doi: <https://doi.org/10.1145/2505341.2505346>
- [13] C. Roberts and J. Kuchera-Morin, "Gibber: Live coding audio in the browser," *Proceedings of the International Computer Music Conference (ICMC)*, Ljubljana, Slovenia, pp.64-69, 2012, <http://hdl.handle.net/2027/spo.bbp2372.2012.011> (accessed Aug. 31, 2026).
- [14] S. Adkins, P. Sarmiento, and M. Barthet, "LooperGP: A loopable sequence model for live coding performance using GuitarPro tablature," *Proceedings of EvoMUSART 2023: Artificial Intelligence in Music, Sound, Art and Design*, Brno, Czech Republic, pp.3-19, 2023.

doi: https://doi.org/10.1007/978-3-031-29956-8_1

- [15] A. Madaan et al., "Self-Refine: Iterative refinement with self-feedback," *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS)*, New Orleans, LA, USA, pp.46534-46594, 2023.

doi: <https://doi.org/10.52202/075280-2019>

- [16] X. Chen, M. Lin, N. Schärli, and D. Zhou, "Teaching large language models to self-debug," *Proceedings of the International Conference on Learning Representations (ICLR)*, Vienna, Austria, 2024.

doi: <https://doi.org/10.48550/arXiv.2304.05128>

- [17] Derivative, TouchDesigner, <https://derivative.ca> (accessed Jul. 12, 2026).

- [18] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," *Proceedings of the 40th International Conference on Machine Learning (ICML)*, Honolulu, HI, USA, pp.28492-28518, 2023.

doi: <https://doi.org/10.48550/arXiv.2212.04356>

저 자 소 개



정 지 응

- 현재 : 전남대학교 문화전문대학원 미디어콘텐츠·컬처테크전공 석사과정
- ORCID : <https://orcid.org/0009-0007-2033-1287>
- 주관심분야 : 오디오비주얼 퍼포먼스, 미디어아트, 생성형 AI



김 한 성

- 연세대학교 전파공학과 학사
- 연세대학교 전기전자공학과 석사
- 연세대학교 전기전자공학과 박사
- 한국과학기술연구원(KIST) 영상미디어센터 연구원
- 일본 國際電氣通信基礎技術研究所 (ATR) 연구원
- 영국 University of Surrey, CVSSP 선임연구원
- 현재 : 영국 University of Southampton 부교수
- ORCID : <https://orcid.org/0000-0003-4907-0491>
- 주관심분야 : 3D Computer Vision, 가상현실



신 춘 성

- 숭실대학교 컴퓨터학부 학사
- 광주과학기술원 정보통신공학과 석사/박사
- 카네기멜론대학 HCI Institute 박사후연구원
- 한국전자기술연구원 VR/AR연구센터 책임연구원
- 문화체육관광부/한국콘텐츠진흥원 문화기술 PD
- 국가과학기술자문회의 ICT-융합 전문위원
- 현재 : 전남대학교 문화전문대학원 미디어콘텐츠·컬처테크전공 교수
- 현재 : 전남대학교 문화기술연구소장
- ORCID : <https://orcid.org/0000-0003-2384-4022>
- 관심분야 : 가상증강현실, HCI, 문화기술