

***Student Name: Junxi Du***

***High School: Beijing No. 80 High School***

***Province: Beijing***

***Country/Region: China***

***Advisor Name: Jinghong Li, Heng Wang***

***Advisor Affiliation: Beijing No. 80 High School***

***Paper Title: Sonara: An Agent-Based Digital Audio Workstation  
for Editable Music Production***

# Sonara: An Agent-Based Digital Audio Workstation for Editable Music Production

*Junxi Du*

## Abstract

Much research in artificial intelligence for music focuses on tasks such as text-to-music generation, melody continuation, and accompaniment generation. However, real music production also involves detailed editing operations, including track management, MIDI editing, and section arrangement. How to enable large language models (LLMs) to understand production intentions and execute these operations requires further systematic study. This paper presents Sonara, an agent-based digital audio workstation (DAW) for music production. The main contributions of this work include: 1) Text-to-Editable Music Production: driven by LLM agents, Sonara translates natural-language or speech instructions into direct operations on tracks, MIDI, instruments, effects, and song structures; 2) DAW100 Dataset: DAW100 provides 100 specified tasks for evaluating editable music-production operations in agent-based DAW systems; 3) Execution-Based Evaluation: Sonara evaluates LLM agents based on the final editable DAW project state, rather than tool-call success alone, jointly measuring task completion, preservation of existing project content, and validity of musical edits. By integrating the agent framework with GPT-5-mini, Sonara achieved an 83% task success rate across two runs of the DAW100 benchmark, demonstrating execution capability across the specified production tasks within an editable DAW environment. A production study with 14 experienced users identified perceived benefits in creative exploration and efficiency, alongside correction needs. Sonara offers a route to reducing operational barriers for producers, with potential applications in accessible music creation through natural language and voice interaction.

**Keywords:** digital audio workstation, language agents, music production, execution evaluation, tool interfaces

## Contents

|                                                                                |    |
|--------------------------------------------------------------------------------|----|
| 1. Introduction .....                                                          | 1  |
| 1.1 From Generating Content to Executing Actions .....                         | 1  |
| 1.2 The Digital Audio Workstation as a Professional Creative Environment ..... | 2  |
| 1.3 The Gap Between Music Generation and Editable Production Control .....     | 2  |
| 1.4 Research Position and Research Questions .....                             | 3  |
| 1.5 Contributions .....                                                        | 3  |
| 2. Related Work .....                                                          | 4  |
| 2.1 Understanding Music Creation Semantics .....                               | 4  |
| 2.2 Agent Execution and Verifiable Task Completion .....                       | 4  |
| 2.3 Accessible Music Production .....                                          | 5  |
| 3. Method: Design and Implementation of Sonara .....                           | 5  |
| 3.1 Overview .....                                                             | 5  |
| 3.2 Rationale for a Self-Built DAW Host .....                                  | 7  |
| 3.3 Four-Layer System Architecture .....                                       | 7  |
| 3.3.1 User Entry Point .....                                                   | 8  |
| 3.3.2 Agent Layer and Tool-Calling Loop .....                                  | 8  |
| 3.3.3 Tool Layer .....                                                         | 9  |
| 3.4 Implementation of the DAW Host .....                                       | 10 |
| 3.5 Readability of Project State .....                                         | 10 |
| 3.6 Example of a Completed Edit .....                                          | 11 |
| 3.7 Implementation Scope .....                                                 | 11 |
| 4. Experimental Design .....                                                   | 12 |
| 4.1 Basis of Automated Evaluation .....                                        | 12 |
| 4.2 Study 1: Execution Capability Across Production Tasks .....                | 13 |
| 4.2.1 DAW100 Construction .....                                                | 13 |
| 4.2.2 Starting Conditions and Execution Protocol .....                         | 14 |
| 4.2.3 Evaluation Metrics .....                                                 | 14 |
| 4.2.4 Model Comparison and Repeated Trials .....                               | 15 |

|                                                                      |    |
|----------------------------------------------------------------------|----|
| 4.2.5 Analysis and Validity .....                                    | 16 |
| 4.3 Study 2: Production Workflow Use .....                           | 16 |
| 4.3.1 Participants and Procedure .....                               | 16 |
| 4.3.2 Measures and Analysis .....                                    | 17 |
| 5. Study 1 Results and Analysis .....                                | 17 |
| 5.1 Overall Results .....                                            | 17 |
| 5.1.1 Overall Execution Performance Across Models .....              | 17 |
| 5.1.2 Condition Satisfaction and Complete Execution .....            | 18 |
| 5.1.3 Performance Across Task Groups .....                           | 18 |
| 5.1.4 Interaction and Tool Use .....                                 | 19 |
| 5.1.5 Consistency Across Repeated Runs .....                         | 21 |
| 5.2 Error Analysis and Interface Implications .....                  | 22 |
| 5.2.1 Condition-Level Performance and Preservation Reliability ..... | 22 |
| 5.2.2 Preserving the Existing Project .....                          | 24 |
| 5.2.3 Recovery Without Task Completion .....                         | 24 |
| 5.2.4 Tool Selection and State Feedback .....                        | 25 |
| 6. Study 2 Results and Analysis .....                                | 26 |
| 6.1 Creativity Support .....                                         | 27 |
| 6.2 Perceived Efficiency and Control .....                           | 28 |
| 6.3 Editing Difficulties and User Requirements .....                 | 29 |
| 7. Discussion .....                                                  | 29 |
| 7.1 Execution Capability in an Editable Project .....                | 29 |
| 7.2 Implications for Tool Interface Design .....                     | 30 |
| 7.3 Creative Control and Workflow Use .....                          | 30 |
| 7.4 Limitations .....                                                | 31 |
| 8. Conclusion .....                                                  | 31 |
| 9. References .....                                                  | 33 |
| Appendix A Execution Settings and Scoring Conventions .....          | 35 |
| A.1 Model and Context Settings .....                                 | 35 |
| A.2 Project Preparation and Shared Rules .....                       | 35 |
| A.3 Scoring Details and Research Records .....                       | 35 |

|                                             |    |
|---------------------------------------------|----|
| Appendix B Detailed Execution Results ..... | 37 |
| Acknowledgement .....                       | 40 |

2026 S.-T. Yau High School Science Award  
仅用于2026丘成桐中学科学奖论文公示

# 1. Introduction

The motivation for Sonara arose from creating background music for school sports clubs. Musical ideas could be developed at the piano, but producing and revising an arrangement required many software operations. A change to one instrumental part could involve locating its track, identifying the relevant clip, and editing individual notes while leaving the rest of the project intact. This practical gap between a production intention and the actions needed to realise it motivated an assistant that operates an editable music environment.



## 1.1 From Generating Content to Executing Actions

Tool use extends the role of large language models (LLMs) from producing responses to performing operations in external systems. A tool specifies an available action, its arguments, and the information returned after execution. Toolformer demonstrated that language models can learn when and how to invoke external APIs (Schick et al., 2023), while ReAct connected action selection with observations obtained from the environment (Yao et al., 2023). Together, these approaches provide a foundation for language agents that carry out instructions through successive interactions with software.

The transition from response generation to execution introduces a distinct requirement: the state of the application must satisfy the requested objective. Selecting a valid tool is insufficient when the operation targets the wrong object, omits a necessary step, or produces an incomplete change. Benchmarks such as OSWorld therefore evaluate outcomes in interactive computer environments using task-specific execution checks (Xie et al., 2024). Music production provides a setting in which these issues can be examined through structured, interdependent edits.

## **1.2 The Digital Audio Workstation as a Professional Creative Environment**

A digital audio workstation (DAW) provides a structured, persistent, and editable environment for recording, composition, arrangement, and sound processing. Tracks organise instrument parts or recorded audio, and clips define regions of material on a timeline. MIDI (Musical Instrument Digital Interface) represents notes through pitch, start time, duration, and velocity (playing intensity); an instrument converts these events into sound. Effects modify the sound, and automation changes parameters during playback. An editable project therefore retains the components and relationships that a rendered audio file alone does not expose.

A production instruction frequently concerns several such objects. For example, adding a piano chord at the beginning of a section may require identifying the section, locating or creating a suitable track, loading an instrument, and inserting notes into a clip. The required sequence depends on the existing project. Correct execution consequently involves musical interpretation, object selection, dependency management, and parameter assignment. Timing conventions are one element of this process, alongside pitch, duration, instrument choice, and effect settings.

## **1.3 The Gap Between Music Generation and Editable Production Control**

Text-conditioned music generation produces audio from descriptions, as exemplified by MusicGen (Copet et al., 2023). Production assistance addresses a related but distinct objective: modifying a composition while retaining control over its constituent parts. A producer may need to revise one instrument, extend a section, or adjust an effect without replacing the surrounding arrangement. An assistant embedded in the DAW can act on these objects and leave the resulting material available for further editing.

MusicAgent coordinates music-processing tools, while DAWZY translates language into reversible actions within REAPER (Elkins et al., 2025; Yu et al., 2023). These precedents

establish the feasibility of language-directed production. The remaining research problem is to connect local editing capabilities with systematic evaluation: whether a request is completed, whether existing material is preserved, and where execution fails. Sonara addresses this problem through an integrated host and tool interface, paired with the DAW100 task set and checks of the resulting project state.

## 1.4 Research Position and Research Questions

This research treats the LLM as an operator of an editable music project. Sonara provides access to track management, note entry, audio arrangement, instruments, effects, automation, and structural markers through named tools. The project remains visible in the graphical interface and available for manual revision. The principal concern is whether language-directed operations realise production intentions, followed by whether users find this interaction useful in practice.

RQ1: Under a fixed Sonara tool interface, how reliably can language models execute production tasks, preserve existing project content, and use tools across different interpretation and operation requirements?

RQ2: How do users with music-production experience perceive Sonara's support for creative exploration, efficiency, and control, and what difficulties arise during use?

Automated project-state checks address RQ1. A production session and questionnaire address RQ2. Together, the studies examine execution reliability and perceived usefulness within the same music-production environment.

## 1.5 Contributions

This paper makes three contributions:

1. **Language-directed editable production.** Sonara connects text and speech to a bounded catalogue of operations within a self-built DAW. Agent actions and manual edits use the same project controllers, keeping the resulting tracks, clips, and notes available for inspection and revision.
2. **DAW100.** A set of 100 production tasks pairs natural-language instructions with prepared starting projects and explicit outcome requirements. The tasks cover direct configuration, musical interpretation, and dependent editing operations.

3. **Project-state evaluation.** An execution-based framework distinguishes requested edits, preservation of existing content, and MIDI validity. Condition-level results and recorded calls support analysis of incomplete edits and concrete requirements for tool-interface design.

## 2. Related Work

### 2.1 Understanding Music Creation Semantics

Music-generation systems and production assistants differ in the representations they manipulate. MusicGen generates audio conditioned on text or melody (Copet et al., 2023). MusicAgent decomposes requests across a collection of music-understanding and generation tools (Yu et al., 2023). A DAW assistant must additionally relate an instruction to the structure of an existing project: the intended track, the material already present, and the relationship between a requested edit and the surrounding arrangement.

DAWZY uses text, voice, and humming input to control REAPER through state queries and generated scripts, including reversible edits (Elkins et al., 2025). Its evaluation includes four reproducible objective tasks, three qualitative comparison tasks, and user ratings. Sonara uses predefined operations within a self-built host and evaluates 100 specified tasks through project-state checks. Its evaluation separates requested changes from preservation and validity, making it possible to identify a correct note pattern placed in an incorrect project structure. The distinction concerns implementation and evaluation coverage; scores from the two task sets are not directly comparable.

### 2.2 Agent Execution and Verifiable Task Completion

Toolformer and ReAct established approaches to selecting external operations and incorporating their outputs into subsequent model behaviour (Schick et al., 2023; Yao et al., 2023). In a persistent editing environment, feedback must also support references to objects created or discovered during earlier actions. A track identifier returned by one operation, for example, may be required by later instrument-loading and clip-editing operations.

OSWorld pairs computer tasks with initial configurations and execution-based evaluation procedures (Xie et al., 2024). Sonara adopts the principle of assessing environmental outcomes within a narrower domain. Its evaluator reads project properties and applies conditions written for each task. This supports reproducible assessment of specified edits, while leaving musical quality and the completeness of the task specification as separate methodological concerns.

Language assistance also interacts with direct manipulation. LAVE executes video-editing objectives through an agent while retaining manual refinement (Wang et al., 2024). A comparable requirement in music is to preserve the editable relationships between tracks, clips, and notes. DAW100 applies execution-based assessment to these relationships, while the production study examines whether users find the resulting interaction useful.

### **2.3 Accessible Music Production**

Accessible music technology encompasses interfaces designed around different abilities and forms of participation. Frid's (2019) review of accessible digital musical instruments provides a foundation for considering access as a relationship between a person's requirements and an interaction design. Sonara focuses on production operations involving repeated pointer actions, including placing clips, entering notes, and adjusting plugin parameters.

Speech can provide an alternative means of requesting production operations, but it also requires users to formulate instructions, initiate input, inspect results, and recover from unwanted changes. These requirements motivate accessible interface design. The present production study examines general workflow experience; it does not establish effectiveness for users with particular motor or input requirements.

## **3. Method: Design and Implementation of Sonara**

### **3.1 Overview**

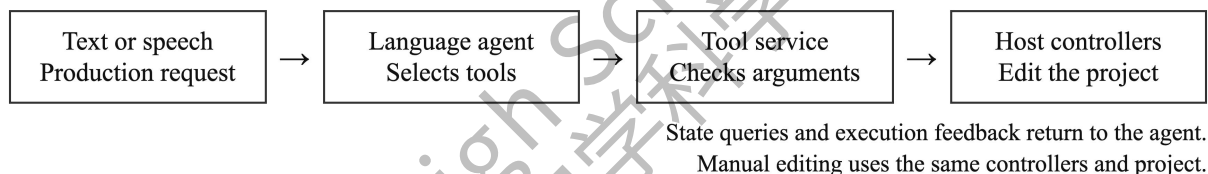
Sonara translates production intentions into changes to a persistent, editable project. Figure 1 connects user input, the agent, the tool service, and the DAW host, then shows how MIDI and recorded audio become audible output. These mechanisms provide the state access and execution path used by DAW100 and its evaluator.

Consider the instruction “add a piano chord at bar 2.” A bar is a group of beats; the current implementation uses four beats per bar. The agent locates or creates a suitable MIDI track and loads a piano instrument if needed. It then locates or creates a clip and inserts the notes forming the chord. The sequence therefore depends on the existing project.

Language control and manual editing converge on the host's editing controllers. For instance, the agent's MIDI-track creation command invokes the controller operation used by the graphical interface. The resulting objects are therefore part of the same project the user inspects and edits. The model does not directly modify host memory or generate the audio signal in this control path.

Figure 2 shows the arrangement interface. The track list identifies the part being edited, while a clip retains its position and extent on the timeline. The annotated view uses a prepared illustration project; the recorded task execution in Figure 4 provides the corresponding evaluation example.

#### A Control and project editing



#### B MIDI and audio rendering paths

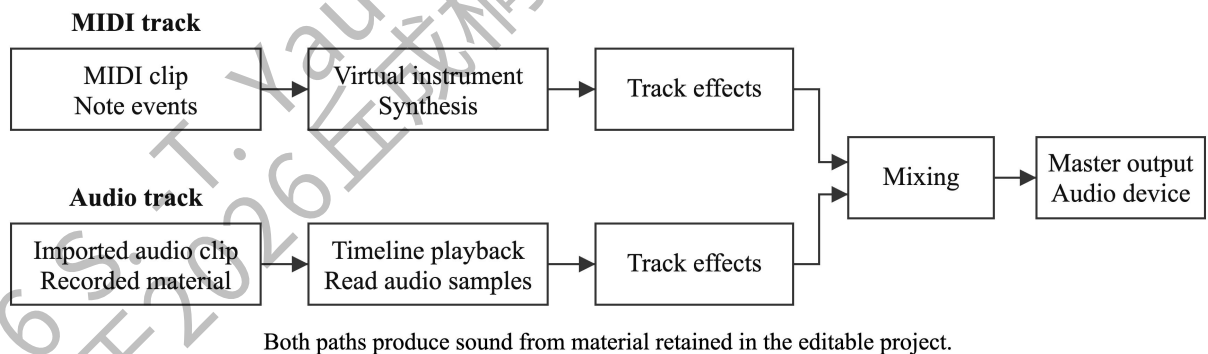


**Figure 1. Project Control and Audio Rendering in Sonara**

*Note.* Panel A summarises control of the editable project. Panel B distinguishes MIDI synthesis from audio-clip playback; both paths pass through track effects and mixing to the output. Manual and agent operations modify the same project.

1 Tempo and transport

2 Track list

3 Editable clip on the timeline

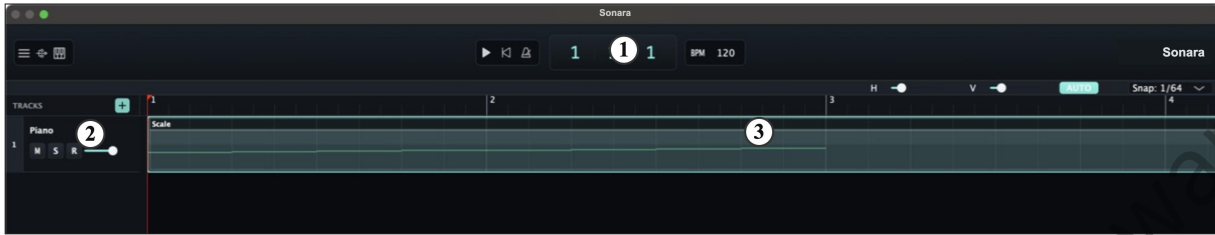


**Figure 2.** Track and Clip Representation in the Sonara Interface

**Note.** Annotated interface view of a prepared illustration project. Labels identify (1) tempo and transport, (2) the track list, and (3) an editable MIDI clip on the timeline. The track is named Piano; its notes form an ascending scale.

### 3.2 Rationale for a Self-Built DAW Host

A self-built host provides control over three aspects of the research environment: the operations exposed to the agent, the properties available to the evaluator, and the interaction mechanisms presented to participants. These components can be revised together as the task coverage and study requirements develop. The architecture also permits inspection of the path from a tool request to the host operation that implements it.

A self-built host requires substantial engineering work and has a smaller feature set than a mature workstation. Sonara provides direct control over editing, interaction, and evaluation within one implementation.

### 3.3 Four-Layer System Architecture

Figure 3 traces a production request through the four layers. Tool results return to the agent as context for its next response, while accepted edits remain available in the host interface.

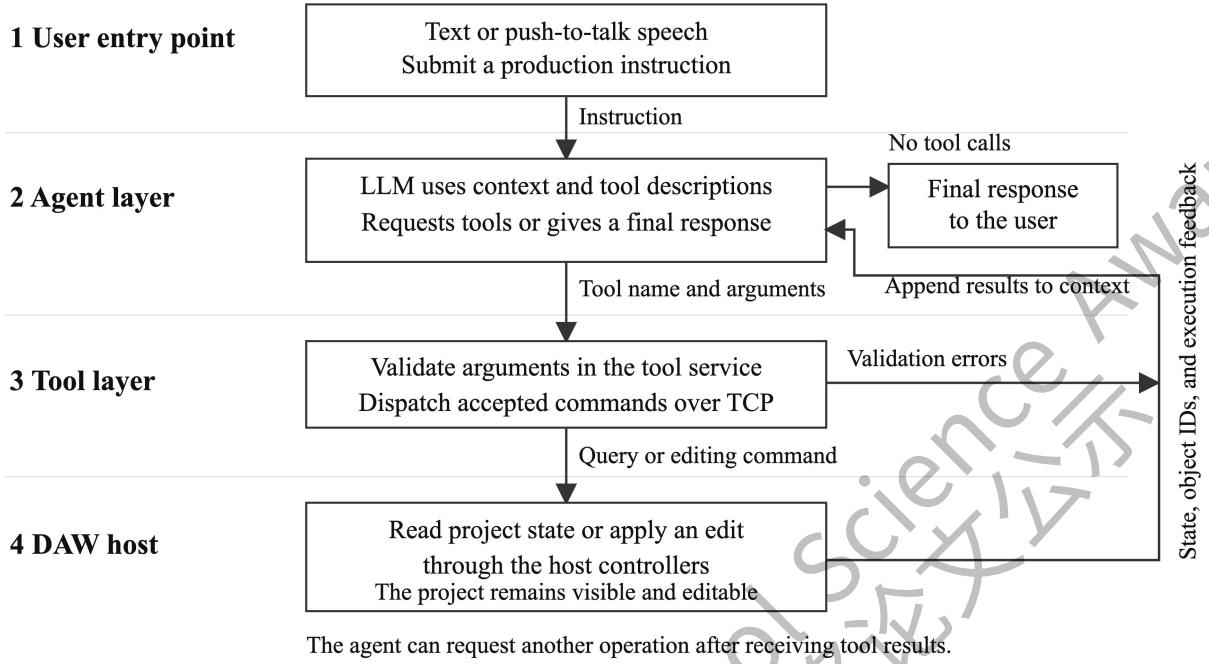


**Figure 3.** Request Execution and Feedback Across the Four System Layers

*Note.* Queries return state without applying an edit. Validation errors also return to the agent. The loop ends when a response contains no tool calls or the 16-round evaluation limit is reached.

### 3.3.1 User Entry Point

The interface accepts typed instructions and push-to-talk speech. Speech interaction supports an integrated backend and a cascaded backend that separates transcription, model reasoning, and speech synthesis. Both paths use the same host editing operations, although their model configurations and communication procedures differ.

Holding the V key starts recording, and releasing it submits the input. Starting another voice input interrupts the active response. This explicit trigger avoids depending on automatic speech detection during music playback, but still requires a physical action. Interruption does not reverse edits already applied to the project.

### 3.3.2 Agent Layer and Tool-Calling Loop

The agent receives the production instruction, conversation context, and descriptions of the available tools. A model response either requests operations or provides a final response. When calls are requested, the controller executes them, appends the returned information to the

conversation, and requests the next model response. State queries and execution feedback can thereby inform subsequent action selection.

Object dependencies constrain this process. A call that requires a newly created track identifier normally follows the response that returns that identifier. Several independent calls can be requested in one round and executed sequentially by the controller. The division into rounds thus reflects both the model's decisions and the information dependencies between operations.

Evaluation permits up to 16 model-response rounds and scores the final project state when the interaction ends. Reaching the limit is recorded separately from task success. A round may contain several tool calls, including attempts to correct earlier errors.

### 3.3.3 Tool Layer

The tool catalogue defines 48 operations, including queries and a help tool. Table 1 summarises the main functional groups. Tools expose editing units such as a batch of notes, a repeated audio clip, or an effect parameter, allowing a production request to be expressed without reproducing every GUI gesture.

*Table 1. Implemented Tool Capabilities*

| Functional group    | Supported operations                                       | Project representation                   |
|---------------------|------------------------------------------------------------|------------------------------------------|
| Tracks and tempo    | Create, rename, and delete tracks; set tempo               | Track structure and timing               |
| MIDI editing        | Create clips; write notes in batches; trim or delete clips | Editable instrument parts                |
| Audio arrangement   | Retrieve local material; import, repeat, and shorten clips | Recorded material on the timeline        |
| Sound processing    | Load instruments and presets; add effects; set parameters  | Instrument and effect chains             |
| Automation and form | Write automation points; define named song sections        | Parameter changes and structural markers |
| State and playback  | Query project objects; control playback                    | Execution feedback and audible output    |

The Python tool service uses the Model Context Protocol to expose operation descriptions and a call mechanism. It translates accepted arguments into commands sent to the host over a local TCP connection. Validation checks argument structure and permitted ranges before dispatch. For example, the batch-note tool accepts pitch, onset, duration, and velocity records, rejecting negative start positions or nonpositive durations. Such checks establish whether an operation is well formed; musical appropriateness depends on the task.

For example, `add_midi_notes_batch` requires a `sequence_id` and an array of note records. Each record contains pitch, clip-relative onset, duration, and velocity. The sequence identifier refers to an existing clip and can be obtained through status. This dependency separates finding the editing target from specifying the note content. The operation appends notes, so correcting an earlier insertion requires attention to material already present.

Note starts are measured relative to their MIDI clip, while clip positions refer to the project timeline. Sonara uses 960 ticks per quarter-note beat and assumes 4/4 time. The tool interface specifies these conventions alongside object identifiers and parameter ranges.

### **3.4 Implementation of the DAW Host**

Sonara is implemented primarily in C++ using JUCE, with Python components providing the tool service and speech bridge. The host coordinates playback and project editing. Its audio-processing graph connects instruments, recordings, and effects to the master output.

Host controllers and message-thread scheduling coordinate project edits. The interface supports further editing but has no project-wide undo for agent operations; unwanted changes require additional edits.

### **3.5 Readability of Project State**

Read-only operations expose tracks, clips, notes, effects, automation, tempo, and song sections. Additional queries retrieve properties such as plugin parameters when a general status response is insufficient. The agent uses this information to identify objects and inspect its progress. An external evaluator can use the same facilities without relying on the assistant's final verbal account.

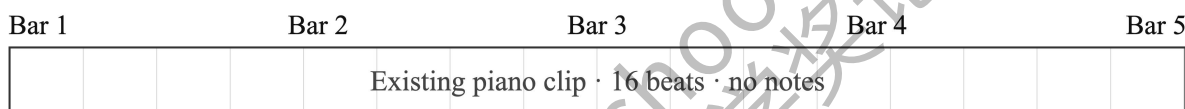
For evaluation, these responses are normalised into a consistent representation. A note's absolute position, for example, is obtained from its clip position and its relative onset. The

representation provides a basis for checking particular requirements while allowing runtime identifiers and unspecified creative choices to vary.

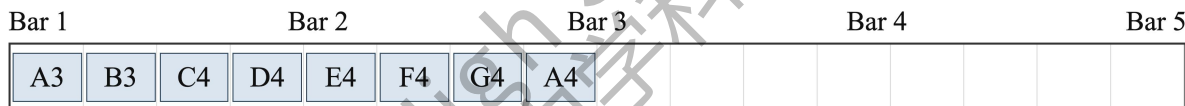
### 3.6 Example of a Completed Edit

Figure 4 illustrates a recorded execution of task L2-10 from Study 1. The instruction requests an ascending A natural minor scale from A3 to A4, starting at bar 1 with one note per beat. The project already contains a piano instrument and an empty four-bar clip. GPT-5-mini retrieves the track, queries project status to locate that clip, and inserts eight notes in one batch. The original clip retains its position and four-bar length; the notes occupy its first two bars. All 13 task checks pass. The example shows how a language request becomes a local edit to an existing project through three tool calls.

#### Before editing



#### After editing



Eight one-beat notes added; clip position and length unchanged.

---

Find the piano track → Locate its existing clip → Insert the note batch

---

Same piano instrument · Same tempo (120 BPM) · 13/13 checks passed

**Figure 4.** A Recorded Edit to an Existing Piano Clip

*Note.* Diagram of recorded project states for GPT-5-mini, first run. Each labelled block is one note lasting one beat. Both rows show the same clip; bar 5 marks its exclusive endpoint. The tool sequence contains the three agent calls, excluding setup and evaluation.

### 3.7 Implementation Scope

The tool set supports track, clip, note, instrument, effect, automation, and section operations. Track volume, panning, and side-chain routing are not exposed. Local-library operations retrieve and arrange existing recordings.

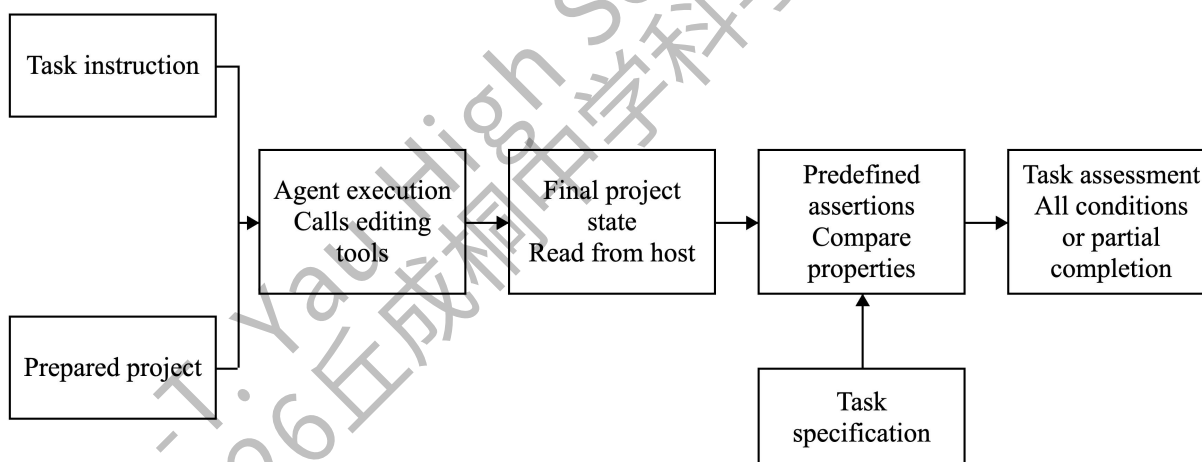
Evaluation uses a common system prompt that excludes demonstration-specific instructions.

## 4. Experimental Design

The evaluation combines automated production tasks and a user study. Study 1 compares six model configurations using DAW100 under a fixed tool interface. Study 2 examines perceived creative support, efficiency, and control through a production session and questionnaire.

### 4.1 Basis of Automated Evaluation

Each task combines an instruction, a prepared starting project, and predefined checks of the required final state. After execution, the evaluator reads the project and applies these checks (Figure 5). The task instruction and shared system prompt specify the requested edits and the properties to preserve.



Examples of checked properties: tempo, track structure, notes, effects, and sections.

**Figure 5.** Task Execution and State-Based Evaluation Procedure

*Note.* Schematic of the evaluation design. Assertions are specified before execution and assess project properties.

Checks assess the requested project properties and allow acceptable alternatives, such as different chord registers or instrument presets. Edits must still target the specified object. Scoring therefore measures execution of the requested changes, rather than perceived musical quality.

## 4.2 Study 1: Execution Capability Across Production Tasks

### 4.2.1 DAW100 Construction

DAW100 contains 100 English production tasks developed with reference to a source collection of 5,745 transcript files from music-related YouTube videos, including production tutorials and practitioner interviews. The texts were obtained by transcribing video audio. Language-model assistance was used to organise and classify source material. Task construction prioritised recurring production topics, support in the Sonara tool interface, and outcomes that could be checked through project state. Instructions were authored in production terminology rather than copied from transcripts or expressed as API commands. The transcripts were not supplied to the evaluated models.

The construction process linked each selected topic to a production request, a starting project, and checks of the expected outcome. For example, L2-02 requests I–V–vi–IV in C on an existing piano track, one chord per bar. Its checks assess C, G, Am, and F together with preservation of the existing clip structure. Side-chain routing, track volume, and panning were excluded because the interface did not expose them; outcomes requiring subjective listening were also excluded. DAW100 therefore covers selected executable operations rather than sampling all production activity in proportion to its frequency.

The tasks cover tempo, tracks, notes, audio placement, instruments, effects, automation, and arrangement sections. Table 2 groups them by the interpretation and operation dependencies they require.

**Table 2.** *DAW100 Task Groups by Interpretation and Operation Requirements*

| Level | Items | Definition                                                            | Example                                                  |
|-------|-------|-----------------------------------------------------------------------|----------------------------------------------------------|
| L1    | 34    | A direct intention with required values stated                        | Set the tempo to 128 beats per minute                    |
| L2    | 38    | Musical interpretation, relative reference, or a state-dependent edit | Write a chord progression on an existing piano track     |
| L3    | 28    | Dependent operations or a target derived from discovered state        | Prepare instrument parts and define arrangement sections |

Before collection, all 100 tasks were checked through reference executions. The evaluator was tested against 224 acceptable states and 1,303 deliberately incorrect variants, and

all 42 exposed tools were exercised. Tasks, software, and scoring rules were then fixed. Development trials were excluded from the reported results. Because earlier model trials informed task refinement, the set is not an independently held-out benchmark.

#### ***4.2.2 Starting Conditions and Execution Protocol***

Each trial begins with a reset and verified project and a fresh model conversation. Starting projects vary by task: an editing task may include a piano track and empty clip, while a construction task may begin without tracks. Models receive the common prompt, task instruction, and any supplied audio-file paths. They must discover project identifiers through queries; reference solutions are withheld.

All models receive the same 42 tools selected from the 48-tool catalogue; six playback-control and presentation tools are excluded (Appendix A). The common prompt specifies how to inspect project state and requires material outside the requested edit to remain unchanged. Preparation and scoring calls are excluded from agent-use measures.

The protocol defines shared timing and editing conventions. Existing clip counts and spans must be preserved unless the task requests a clip edit. Equivalent chord registers, inversions, and specified instrument alternatives are accepted. Appendix A provides the detailed conventions and execution settings.

#### ***4.2.3 Evaluation Metrics***

Evaluation follows three requirements of local production editing: realise the requested change, retain material outside its scope, and leave valid project data. Task-level success tests whether these requirements hold together. Condition-level scores show which parts are satisfied when a task remains incomplete. These measures apply established execution-based evaluation principles to the objects and constraints of an editable DAW project.

For example, a four-chord task requires C, G, Am, and F on an existing piano track, with each chord starting at the next bar and lasting one bar. A joint check assesses note pitch, onset, and duration while allowing equivalent registers and inversions. Separate checks assess preservation of the existing clip structure. Numerical properties use predefined tolerances; specified pitches and counts use exact matching.

Task success rate (TSR), the primary outcome, is the proportion of trials passing every check. Condition satisfaction rate (CSR) is the proportion of individual checks passed. Each 100-

task run contains 165 direct-request checks, 956 preservation checks, and 100 MIDI-validity checks. These assess the requested edits, unchanged project properties, and valid note placement and values, respectively. All CSR includes all three categories; Direct CSR includes only requested edits. Tasks with more checks contribute more to CSR, and a joint check counts as one scoring unit. Figure 6 illustrates how the three rates differ.

| Task              | Direct requests |                    |      |      | Preservation |                        |      | Validity |      |
|-------------------|-----------------|--------------------|------|------|--------------|------------------------|------|----------|------|
|                   | D1              | D2                 | D3   | D4   | P1           | P2                     | P3   | P4       | V1   |
| Task A            | Pass            | Pass               | Pass | Pass | Pass         | Pass                   | Pass | Pass     | Pass |
| Task B            | Pass            | Pass               | Fail | Fail | Pass         | Pass                   | Pass | Pass     | Pass |
|                   |                 |                    |      |      |              |                        |      |          |      |
| TSR               |                 | Direct CSR         |      |      |              | All CSR                |      |          |      |
| 1 / 2 tasks = 50% |                 | 6 / 8 checks = 75% |      |      |              | 16 / 18 checks = 88.9% |      |          |      |

One column is one check. Only Task A passes every check.

**Figure 6.** Illustrative Calculation of Task and Condition Satisfaction

*Note.* Hypothetical example, not experimental data. Each task has four direct-request checks (D1–D4), four preservation checks (P1–P4), and one validity check (V1). Task B fails two direct requests. Actual tasks contain different numbers of checks.

Process measures are response rounds, requested tool calls, and tool coverage. A round is one model response, including a final text response; it may request several tools. Calls count individual tool requests, including unsuccessful calls within scored trials. Coverage is the number of exposed tools returning successfully at least once across both runs, out of 42. Preparation, scoring, and infrastructure-failure attempts are excluded from these measures.

#### 4.2.4 Model Comparison and Repeated Trials

The comparison includes GPT-4.1-nano, GPT-4o-mini, GPT-4.1-mini, GPT-4.1, GPT-5-nano, and GPT-5-mini, using fixed API snapshots. GPT-4.1 and GPT-4o configurations use temperature 1.0; GPT-5 configurations use medium reasoning effort. All models have the same response-token limit. Appendix A lists the snapshots and settings.

Models receive identical tool schemas and a shared prompt. Tool descriptions and feedback are length-limited in the model context; complete host responses are retained in the logs.

Each of the six models completed two runs of the same 100 tasks, yielding 1,200 scored trials. Models share the same randomised task order within each repetition, and model order is also randomised. Trials execute one at a time on a dedicated host to keep project states separate.

#### ***4.2.5 Analysis and Validity***

The primary comparison combines both runs, giving 200 trials per model. Overall and task-group TSR and CSR describe performance; 95% TSR intervals use 4,000 bootstrap samples of task IDs, retaining both repetitions of each sampled task. Repeatability compares success rates and matched task outcomes across the two runs for all six models. Checks and repeated executions are not independent task samples.

Error analysis links failed checks to tool calls, feedback, and final project states. It examines requested content, project preservation, recovery, and tool selection. All GPT-5-nano trials reaching the round limit are reviewed and compared with the same task and repetition for GPT-5-mini. Detailed examples use first-run traces unless another repetition is specified.

Only trials with an exported and scored final state enter TSR and CSR. Infrastructure interruptions are recorded separately, and scored model failures are not rerun to select a better result.

Logs preserve model interactions, tool responses, final project states, and individual check outcomes. Archived task and software configurations support score replay and verification.

### **4.3 Study 2: Production Workflow Use**

#### ***4.3.1 Participants and Procedure***

Fourteen participants with music-production experience completed a 20-minute free-creation session using Sonara with GPT-5-mini, followed by a questionnaire. Text input, speech input, and manual editing were permitted, and on-site guidance was provided. Exact years of DAW experience were collected separately from the questionnaire categories. Mean DAW experience was 4.1 years, and nine participants reported prior use of AI music tools. The questionnaire addressed creative support, perceived efficiency, the intuitiveness of semantic instructions, correction needs, and trust in agent operations. Open-ended responses described difficulties and desired improvements.

### **4.3.2 Measures and Analysis**

The questionnaire includes 11 items addressing exploration, expressiveness, immersion, enjoyment, results worth effort, and collaboration, dimensions associated with the Creativity Support Index (CSI; Cherry & Latulipe, 2014). Items use a five-point agreement scale, from 1 (strongly disagree) to 5 (strongly agree). Results describe all six dimensions rather than an overall CSI score. For enjoyment and exploration, the two item scores are averaged for each participant; the reported mean and standard deviation are then calculated across the 14 participant averages. Results worth effort averages its two items across all 14 participants. Immersion uses the two item means separately; collaboration is assessed with one item about creating with other people. System-specific responses are summarised as participant counts and percentages. The researcher read the open-ended responses and grouped them descriptively into three themes: musical commands, editing existing material, and recovery. Efficiency and trust are assessed as participant perceptions. The original trust item referred to both visible operations and reversibility. The revised questionnaire refers only to visibility; the results reported here derive from the original item.

## **5. Study 1 Results and Analysis**

Study 1 comprises 1,200 scored trials: six model configurations, 100 tasks, and two runs. Overall results and condition-level analyses combine both runs. A separate comparison reports agreement between the two executions of each task.

### **5.1 Overall Results**

#### **5.1.1 Overall Execution Performance Across Models**

GPT-5-mini achieved the highest TSR at 83.0%, followed by GPT-4.1 at 72.0% (Figure 7). GPT-4.1-mini and GPT-5-nano produced similar aggregate results, at 62.5% and 61.0%, respectively. GPT-4o-mini achieved 48.0%, and GPT-4.1-nano achieved 35.0%. Each rate uses 200 trials. Table B1 gives condition-satisfaction rates and confidence intervals.

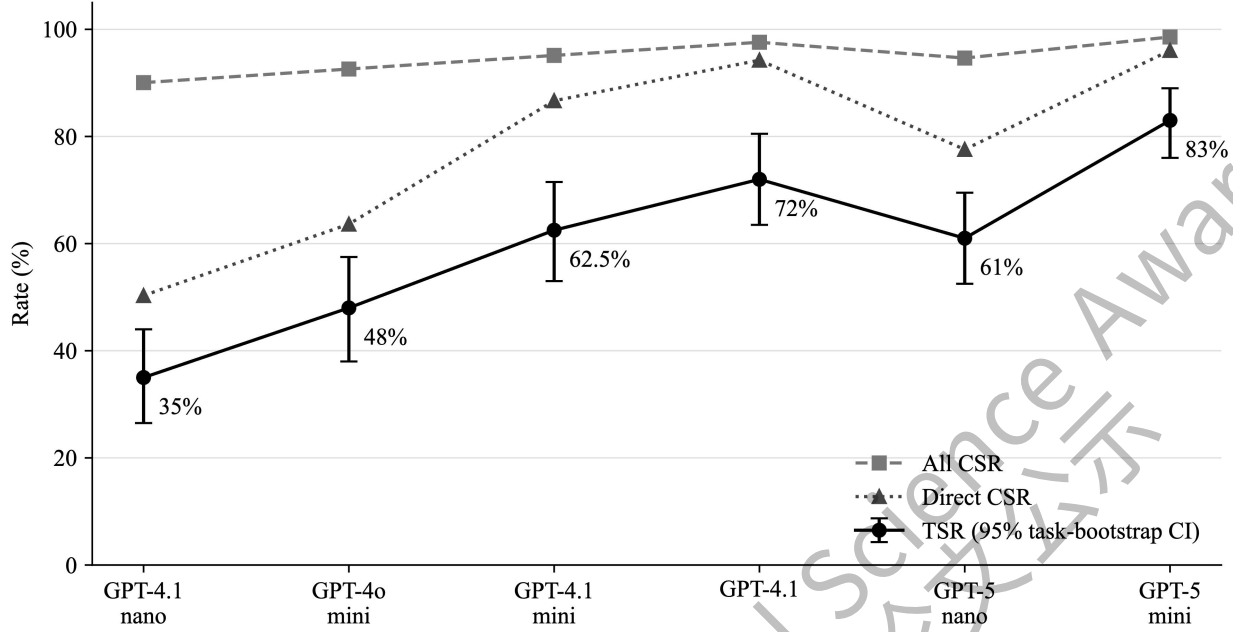


**Figure 7. Condition Satisfaction and Task Success Across Models**

*Note.* Each model completed 100 tasks twice. Error bars show 95% task-bootstrap intervals for TSR, retaining both runs together. All CSR uses 2,442 checks per model; Direct CSR uses 330 direct-request checks.

### 5.1.2 Condition Satisfaction and Complete Execution

All CSR ranged from 90.0% to 98.6%, while Direct CSR ranged from 50.3% to 96.1%. Across both runs, the 2,442 checks per model comprise 330 direct-request checks, 1,912 preservation checks, and 200 MIDI-validity checks.

The large share of preservation checks helps explain why pooled CSR remained high even for models with limited task completion. Properties not affected by an unsuccessful edit can still satisfy those checks. Direct-request CSR therefore complements the pooled measure. GPT-5-mini satisfied 98.6% of all checks but completed 83.0% of trials, while GPT-4.1 achieved 97.6% and 72.0%. Correctness on most properties is insufficient when a task requires all conditions to hold together.

### 5.1.3 Performance Across Task Groups

Differences were more pronounced in editing and combined-operation tasks than in basic operations. GPT-5-mini and GPT-4.1 achieved similar L1 success rates, at 64/68 (94.1%) and 66/68 (97.1%). In L2, their results were 63/76 (82.9%) and 42/76 (55.3%). In L3, they completed

39/56 (69.6%) and 36/56 (64.3%). The largest gap occurred in L2, where GPT-5-mini completed 21 more trials.

L1–L3 group tasks by interpretation and operation dependencies, rather than an empirically calibrated difficulty scale. GPT-4.1 performed better on the tasks represented in L3 than on those in L2. The reversal shows that task-group success also depends on the particular operations and requests in each group (Figure 8).

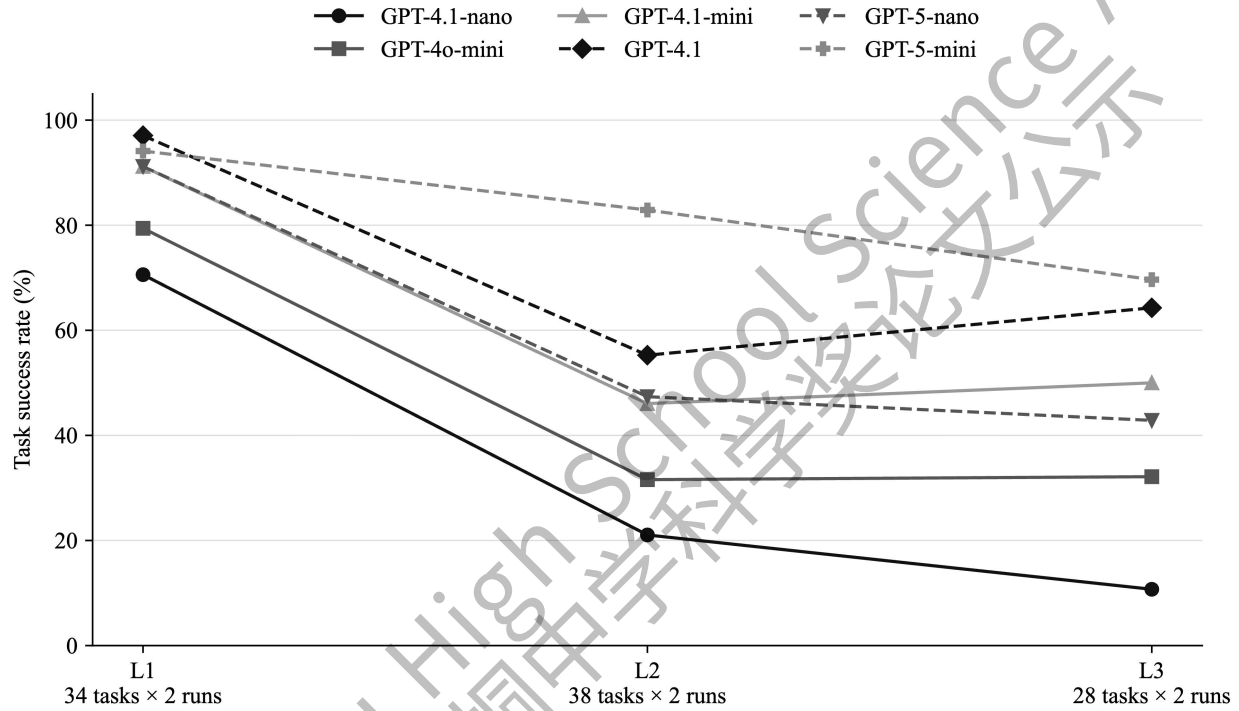


**Figure 8.** Task Success Across Production Task Groups

*Note.* Each model completed two runs of 34 L1, 38 L2, and 28 L3 tasks, giving 68, 76, and 56 trials, respectively.

#### 5.1.4 Interaction and Tool Use

Interaction counts did not follow the same ordering as task success (Table 3). GPT-4.1 had the lowest mean rounds and calls, at 3.60 and 3.02 per trial. GPT-5-mini achieved the highest TSR with 4.42 rounds and 3.53 calls. GPT-5-nano used the most interaction, averaging 5.84 rounds and 5.33 calls, and reached the 16-round limit in 20 trials. GPT-4.1 never reached the limit; GPT-5-mini reached it once.

**Table 3.** Interaction Counts and Tool Coverage Across Both Runs

| Model        | Rounds<br>M (SD) | Calls<br>M (SD) | Limit<br>reached | Tool<br>coverage |
|--------------|------------------|-----------------|------------------|------------------|
| GPT-4.1-nano | 5.50<br>(3.07)   | 4.88<br>(3.28)  | 3                | 34/42<br>(81.0%) |
| GPT-4o-mini  | 4.14<br>(2.89)   | 3.62<br>(3.20)  | 8                | 24/42<br>(57.1%) |
| GPT-4.1-mini | 3.85<br>(1.98)   | 3.17<br>(2.18)  | 2                | 30/42<br>(71.4%) |
| GPT-4.1      | 3.60<br>(1.45)   | 3.02<br>(1.76)  | 0                | 28/42<br>(66.7%) |
| GPT-5-nano   | 5.84<br>(4.45)   | 5.33<br>(4.76)  | 20               | 32/42<br>(76.2%) |
| GPT-5-mini   | 4.42<br>(2.20)   | 3.53<br>(2.43)  | 1                | 28/42<br>(66.7%) |

**Note.** Means and standard deviations use 200 scored trials per model and include unsuccessful tool calls. Limit reached counts trials ending at the round cap. Coverage is the union of tools returning successfully across both runs.

Figure 9 relates task success to interaction counts. GPT-5-mini completed more tasks than GPT-4.1 but used more rounds and calls on average. GPT-5-nano used more interaction than either model without higher success. More interaction can support dependent operations or correction, but can also accumulate without resolving an error; its value therefore depends on what the calls accomplish.

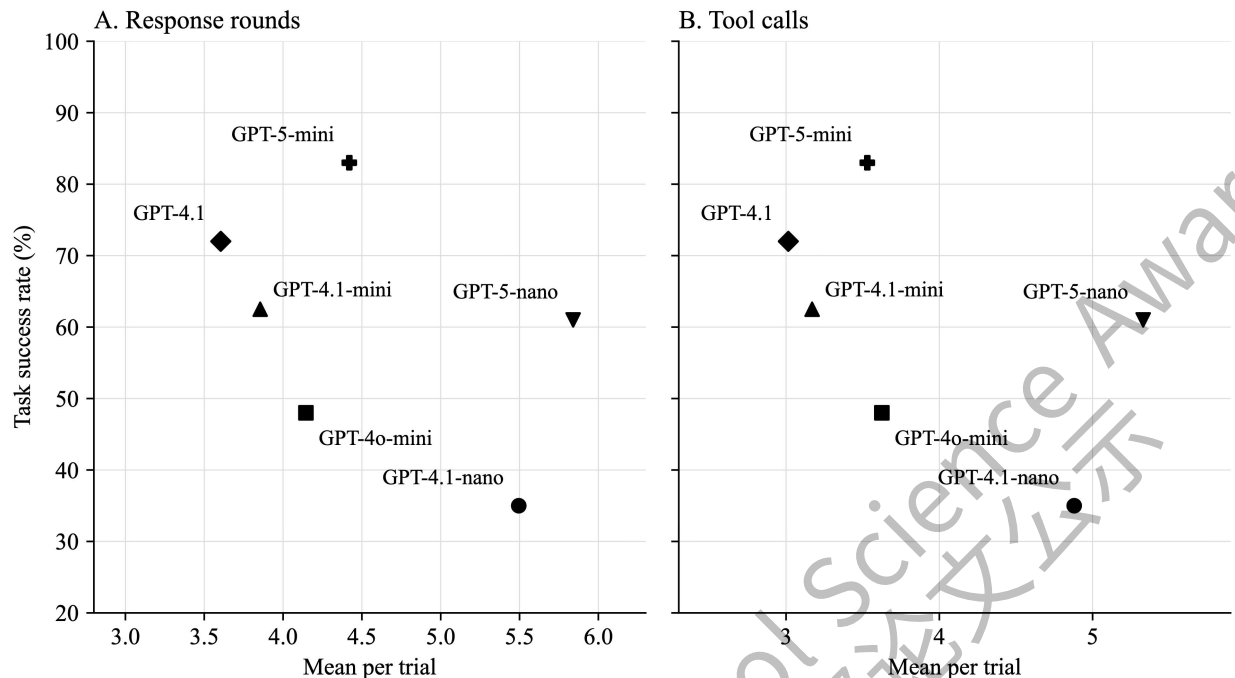


**Figure 9.** Task Success in Relation to Interaction Counts

**Note.** Each point represents one model across 200 trials. Means include unsuccessful calls. Panel A uses response rounds; Panel B uses individual tool calls.

Across all six models, mean rounds and calls increased from L1 to L2 to L3 (Table B2). GPT-5-nano rose from 3.15 rounds and 2.22 calls in L1 to 8.04 rounds and 8.27 calls in L3. GPT-5-mini rose from 3.16 rounds and 2.21 calls to 6.25 rounds and 5.54 calls. In L2, GPT-5-mini completed 63/76 trials with 3.24 calls on average, compared with 42/76 and 3.05 calls for GPT-4.1. GPT-5-mini's higher success accompanied a modest increase in calls. These comparisons describe execution effort, not monetary cost or token use.

Tool coverage describes breadth of use rather than interaction effort. GPT-4.1-nano successfully invoked 34 of the 42 exposed tools and GPT-5-nano invoked 32, compared with 28 for GPT-5-mini. A successful tool return can occur within an unsuccessful task, so wider coverage did not imply better task execution.

### 5.1.5 Consistency Across Repeated Runs

Success rates differed by 0–4 percentage points between runs, but task-level agreement ranged from 82% to 96% (Table 4). GPT-5-nano remained at 61% while outcomes changed on 18 tasks: nine changed from success to failure and nine in the opposite direction. GPT-5-mini remained at

83%, but outcomes changed on 12 tasks; 77 tasks succeeded twice. Similar overall scores therefore concealed variation in which tasks were completed.

**Table 4.** Task Success and Agreement Across Two Runs

| Model        | Run 1<br>TSR (%) | Run 2<br>TSR (%) | Task agreement<br>(%) |
|--------------|------------------|------------------|-----------------------|
| GPT-4.1-nano | 33               | 37               | 92                    |
| GPT-4o-mini  | 48               | 48               | 96                    |
| GPT-4.1-mini | 62               | 63               | 95                    |
| GPT-4.1      | 71               | 73               | 94                    |
| GPT-5-nano   | 61               | 61               | 82                    |
| GPT-5-mini   | 83               | 83               | 88                    |

*Note.* Each run contains the same 100 tasks. Agreement counts tasks that either passed in both runs or failed in both runs.

## 5.2 Error Analysis and Interface Implications

### 5.2.1 Condition-Level Performance and Preservation Reliability

Basic configuration requirements were generally well satisfied. Tempo changes passed 107/108 checks (99.1%). Track type, track identity and count, and section arrangement passed 177/180 (98.3%), 161/168 (95.8%), and 166/180 (92.2%), respectively. Sections are named timeline regions such as Intro or Verse, distinct from clips containing notes or audio. The joint note-pattern check passed 220/408 times (53.9%). It requires the specified pitches, onsets, durations, and note count together. Results ranged from 0/68 for GPT-4.1-nano to 65/68 for both GPT-4.1 and GPT-5-mini. Note execution therefore distinguished models, rather than representing a uniformly weak capability.

Automation changes a parameter during playback, such as an effect setting across a section. Requested automation passed 14/36 checks (38.9%). GPT-4.1-mini and GPT-5-nano passed 4/6 each, while GPT-4.1 and GPT-5-mini passed 3/6 each. These counts cover only three distinct automation checks per model, each repeated twice. MIDI validity was higher at 1,157/1,200 (96.4%), but valid note placement and values can remain satisfied even when the requested pattern is absent.

Preservation showed a more consistent weakness. Overall preservation rates ranged from 96.6% to 98.8%, yet preservation of clip counts, positions, and lengths passed only 660/912 checks (72.4%). With 152 checks per model across both runs, rates ranged from 62.5% to 88.2%; this was the lowest-performing preservation type for every model. It accounted for 252 of 291 failed preservation checks (86.6%), while every other preservation type exceeded 98% when pooled across models. Figure 10 summarises these condition types.

#### A Selected requested changes

| Requirement            | GPT-4.1 nano    | GPT-4o mini     | GPT-4.1 mini    | GPT-4.1         | GPT-5 nano      | GPT-5 mini      |
|------------------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| Tempo                  | 94.4%<br>17/18  | 100.0%<br>18/18 | 100.0%<br>18/18 | 100.0%<br>18/18 | 100.0%<br>18/18 | 100.0%<br>18/18 |
| Track type             | 100.0%<br>30/30 | 100.0%<br>30/30 | 93.3%<br>28/30  | 100.0%<br>30/30 | 96.7%<br>29/30  | 100.0%<br>30/30 |
| Track identity / count | 100.0%<br>28/28 | 92.9%<br>26/28  | 92.9%<br>26/28  | 100.0%<br>28/28 | 89.3%<br>25/28  | 100.0%<br>28/28 |
| Section arrangement    | 73.3%<br>22/30  | 100.0%<br>30/30 | 96.7%<br>29/30  | 100.0%<br>30/30 | 83.3%<br>25/30  | 100.0%<br>30/30 |
| Instrument             | 71.9%<br>23/32  | 68.8%<br>22/32  | 71.9%<br>23/32  | 87.5%<br>28/32  | 87.5%<br>28/32  | 93.8%<br>30/32  |
| Track effects          | 56.7%<br>17/30  | 76.7%<br>23/30  | 100.0%<br>30/30 | 90.0%<br>27/30  | 93.3%<br>28/30  | 100.0%<br>30/30 |
| Audio placement        | 28.9%<br>11/38  | 68.4%<br>26/38  | 92.1%<br>35/38  | 92.1%<br>35/38  | 76.3%<br>29/38  | 97.4%<br>37/38  |
| Note pattern           | 0.0%<br>0/68    | 10.3%<br>7/68   | 75.0%<br>51/68  | 95.6%<br>65/68  | 47.1%<br>32/68  | 95.6%<br>65/68  |
| Clip structure         | 0.0%<br>0/14    | 0.0%<br>0/14    | 71.4%<br>10/14  | 100.0%<br>14/14 | 71.4%<br>10/14  | 100.0%<br>14/14 |
| Automation             | 0.0%<br>0/6     | 0.0%<br>0/6     | 66.7%<br>4/6    | 50.0%<br>3/6    | 66.7%<br>4/6    | 50.0%<br>3/6    |

#### B Preservation of existing material

|                  |                    |                    |                    |                     |                    |                    |
|------------------|--------------------|--------------------|--------------------|---------------------|--------------------|--------------------|
| Clip structure   | 66.4%<br>101/152   | 68.4%<br>104/152   | 62.5%<br>95/152    | 73.7%<br>112/152    | 75.0%<br>114/152   | 88.2%<br>134/152   |
| Other properties | 99.7%<br>1754/1760 | 99.7%<br>1755/1760 | 99.5%<br>1752/1760 | 100.0%<br>1760/1760 | 99.1%<br>1744/1760 | 99.8%<br>1756/1760 |

Darker cells indicate higher satisfaction. Counts combine both runs of each task.

**Figure 10. Condition Satisfaction Across Models and Requirement Types**

**Note.** Each cell reports the percentage and passed/total checks for one model across both runs. Panel A contains ten selected direct-requirement types; Panel B separates clip-structure preservation from all other preservation checks.

Darker cells indicate higher satisfaction. Counts include repeated checks of the same tasks.

The condition profiles show that aggregate rankings do not apply uniformly to every operation (Figure 10). GPT-5-mini passed 37/38 audio-placement checks and 65/68 note-pattern checks, compared with 29/38 and 32/68 for GPT-5-nano. GPT-4.1 matched GPT-5-mini on the note-pattern check despite lower overall task success. These differences support examining particular requirements alongside the aggregate outcome.

The joint note check identifies an incorrect pattern but does not, by itself, identify the cause. In L2-10, GPT-5-nano was asked for an ascending A natural minor scale. Its first batch repeated only two pitches instead of the eight successive pitches of the scale. The call was accepted because the note records were well formed, while the content check failed. This separates argument validity from musical correctness and motivates inspecting the complete note pattern against the request.

### ***5.2.2 Preserving the Existing Project***

The successful edit in Figure 4 provides a matched comparison for L2-10. GPT-4.1 also inserted the correct scale in three calls, but created a new eight-beat clip alongside the existing 16-beat clip. The notes passed, while the additional clip violated the preservation requirement. GPT-5-mini instead used a status query to locate the existing clip. Equal call counts therefore concealed a consequential difference in object discovery and selection.

The same distinction remained relevant to the best-performing model. In L2-02, GPT-5-mini produced the correct four-chord progression but placed it in a new clip alongside the existing one. A separate failure involved exceeding the requested scope: in L1-25, it created the requested four-bar piano clip and then added 12 unsolicited notes. It subsequently inspected those notes and described the added progression in its final response. Inspection alone therefore did not ensure compliance; the resulting state also had to be compared with the requested scope.

### ***5.2.3 Recovery Without Task Completion***

All 20 GPT-5-nano trials reaching the 16-round limit failed: nine in L2 and eleven in L3. Eighteen contained no explicit tool-error response, and sixteen included clip deletion and recreation. At scoring, seventeen trials failed note-pattern checks, twelve failed clip-structure checks, seven failed track-identity/count checks, two failed MIDI-validity checks, two failed section-arrangement checks, one failed an instrument check, and one failed an effects check.

These categories overlap. Table B3 summarises every capped trial and the matched GPT-5-mini outcome from the same repetition.

The first-run traces show distinct forms of incomplete execution. In L2-10, GPT-5-nano repeatedly submitted the same incorrect pitch pattern while deleting and rebuilding clips; it created five clips and deleted three. GPT-5-mini instead queried the existing clip and inserted the correct scale in three calls. In L3-03, GPT-5-nano produced the correct four chords but spread them across four clips when one was required. It reached the limit through continued construction and inspection, rather than repeated rejection of a call. GPT-5-mini used one 16-beat clip and a single note batch, completing the task in seven calls.

GPT-5-mini completed ten of the twenty matched trials. The remaining failures show that lower aggregate interaction counts do not ensure compliance. In the second-run L3-10 trace, GPT-5-nano repeatedly added, cleared, and removed effects without satisfying the required effect configuration. The capped traces support checking progress against the complete request before repeating an edit; they do not establish that a larger round budget would resolve the failures.

#### 5.2.4 Tool Selection and State Feedback

Across both runs of all six models, 40 of the 42 exposed tools were attempted and 39 returned successfully at least once. Two tools were never called; one was attempted without a successful return (Table 5). No task directly requested instrument removal or a plug-in count, and supplied media were audio files rather than MIDI files. These gaps limit conclusions about tool selection: lack of use does not by itself show that a tool is unnecessary or poorly designed.

**Table 5.** *Tools Without a Successful Return Across Both Runs*

| Tool              | Attempted calls | Successful returns |
|-------------------|-----------------|--------------------|
| get_plugin_count  | 0               | 0                  |
| remove_instrument | 0               | 0                  |
| load_midi_file    | 3               | 0                  |

*Note.* Counts cover agent calls in all 1,200 scored trials.

The unsuccessful MIDI import illustrates a selection error. In L2-23, GPT-4.1-nano passed the supplied WAV audio file to load\_midi\_file, which rejected the format. The call selected the wrong operation for the supplied media type. Existing format validation rejected the

call; making supported extensions explicit in the model-visible schema may improve tool selection.

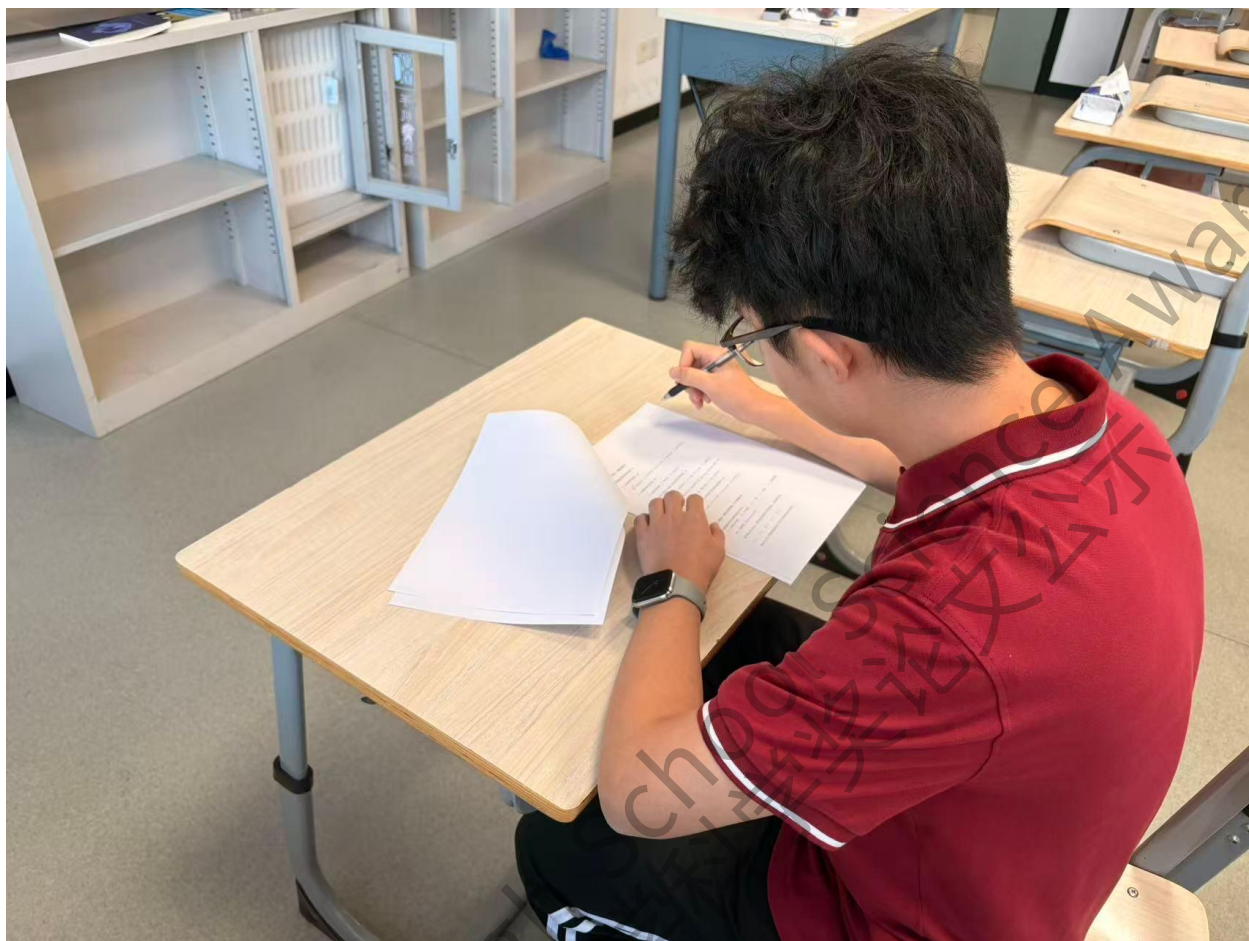
The preservation cases provide a more concrete interface concern. The observed `get_all_tracks` response supplied track identifiers but omitted MIDI clip identifiers and spans. These were available through status, whose brief tool description referred only to playback state and position. The system prompt explicitly directed models to status for clip IDs, so the information was accessible; nevertheless, the tool description did not make this role clear. Returning clip summaries with track information and identifying state inspection clearly in the tool description may make existing clips easier to locate.

In L3-18, the request combined a tempo change, a piano instrument, and a chord progression. GPT-5-nano corrected a malformed note array after an explicit error, but left earlier unwanted notes in place and never loaded the piano instrument. Repeated `add_midi_notes_batch` responses used `notes_count` for the submitted batch size; a later `get_midi_notes` query revealed 14 accumulated notes in the earlier clip. Successful insertion feedback therefore described the local operation without clearly distinguishing the batch from the resulting total. Returning both counts would make accumulation easier to detect.

The exposed interface also lacked a dedicated operation to update or delete selected notes, making some corrections dependent on clip reconstruction. These traces identify three concrete requirements: explicit clip summaries, feedback describing the state after insertion, and note-editing operations that preserve clip metadata. Their purpose is to make the target and effects of an edit easier to verify; the present evaluation does not measure the benefit of implementing these changes.

## 6. Study 2 Results and Analysis

Fourteen participants completed the production session and questionnaire. Table 6 summarises the reported creative-support ratings. Figure 11 presents system-specific responses, with all participant proportions using a denominator of 14.



### 6.1 Creativity Support

Enjoyment ( $M = 4.2$ ,  $SD = 0.6$ ), exploration ( $M = 4.1$ ,  $SD = 0.7$ ), and expressiveness ( $M = 4.0$ ,  $SD = 0.7$ ) received the highest mean ratings. Results worth effort averaged 3.9 ( $SD = 0.8$ ), followed by collaboration ( $M = 3.5$ ,  $SD = 1.0$ ). The two immersion-item means were 3.1 and 3.4. These ratings describe perceived creative support rather than an independent assessment of musical output quality.

**Table 6.** *Reported Creative-Support Ratings After Production Use*

| Dimension            | Reported mean | SD  |
|----------------------|---------------|-----|
| Enjoyment            | 4.2           | 0.6 |
| Exploration          | 4.1           | 0.7 |
| Expressiveness       | 4.0           | 0.7 |
| Results worth effort | 3.9           | 0.8 |
| Collaboration        | 3.5           | 1.0 |
| Immersion            | 3.1–3.4       | —   |

**Note.**  $N = 14$ . Ratings use a five-point agreement scale. Enjoyment and exploration use participant-level averages of two items. Results worth effort averages two items across participants; collaboration uses one item. Immersion gives the range of two item means, with no *SD* reported. No overall CSI score is calculated.

## 6.2 Perceived Efficiency and Control

Ten participants (71.4%) reported a moderate or large improvement in efficiency, and ten (71.4%) rated semantic instructions as intuitive or very intuitive. On the original trust item, ten participants (71.4%) reported increased trust, including four (28.6% of the full sample) who reported a large increase. These counts concern the original wording about visibility and reversibility. These responses describe perceived efficiency, intuitiveness, and trust in the described operations, rather than measured changes in completion time or a causal effect on trust.

### Participant reports after production use

14 participants; categories may overlap



### Reported difficulty



**Figure 11.** Participant Reports of Efficiency, Intuitiveness, Trust, and Correction

**Note.** Counts refer to the same 14 participants and may overlap. Correction frequency is a participant report, not a tool-call error rate. Trust counts derive from the original item, which referred to visibility and reversibility, rather than the revised visibility-only item.

## 6.3 Editing Difficulties and User Requirements

Eleven participants (78.6%) reported at least occasional correction cycles caused by misinterpretation. Open-ended responses highlighted three requirements: higher-level musical commands, such as changing a progression to minor; editing an existing clip instead of creating another one; and broader undo coverage for agent actions. The clip-editing concern is consistent with the preservation failures in Study 1, while the request for undo identifies the practical importance of recovering from an unwanted change.

## 7. Discussion

### 7.1 Execution Capability in an Editable Project

Sonara demonstrates language-directed editing through an integrated host, while DAW100 makes the resulting operations assessable against explicit requirements. With GPT-5-mini, the system completed 166 of 200 trials (83.0%). GPT-4.1 and GPT-5-mini had similar results on direct L1 operations, but GPT-5-mini completed 63/76 L2 trials compared with 42/76 for GPT-4.1. Their L3 results were 39/56 and 36/56. These observations answer RQ1 for the tested configurations and show where interpretation and state-dependent editing differentiate them.

They assess execution of production instructions, not the artistic quality of an original composition.

A correct local edit has two requirements: the requested material must be produced in the intended place, and material outside the request must remain intact. The clip failures show why evaluating only new notes would overstate completion. Conversely, a high pooled condition score can be dominated by unchanged properties even when the edit fails. DAW100 makes these requirements explicit through separate checks. Its value is therefore both comparative and diagnostic: it locates the specific part of an editable project that prevents a task from being complete.

## **7.2 Implications for Tool Interface Design**

The traces and participant responses identify four interface requirements. Object discovery should make existing tracks and clips easy to identify. Local editing should support changing selected notes while preserving clip metadata. Feedback should distinguish the operation performed from the resulting project state, including accumulated content. Recovery should provide a practical way to reverse unwanted edits. The first three follow from observed execution paths; the production study adds a direct request for broader undo coverage.

The self-built host connects editing controllers, state queries, and task checks, making these relationships inspectable. A tool can accept a call without the project satisfying the user's request, and querying the result is useful only if the retrieved state is compared with that request. DAW100 provides concrete examples of this verification gap. The resulting design requirements concern how the interface exposes and modifies project state; they do not depend solely on replacing the language model.

## **7.3 Creative Control and Workflow Use**

Study 2 addresses RQ2 through reported experience: ten participants perceived an efficiency benefit, while eleven reported at least occasional correction needs. These perceptions do not establish faster completion or reliable control. Enjoyment and exploration exceeded the immersion-item means, suggesting that enjoyment need not imply absorption. Expressiveness exceeded collaboration, which concerned creating with other people. Language assistance should

therefore be assessed alongside the effort needed to formulate requests, inspect edits, and correct unwanted changes.

Both studies highlight preservation of the surrounding project. Automated traces show correct notes accompanied by unwanted clip changes; participants likewise requested edits to existing clips. This convergence supports the practical relevance of preservation checks and feedback that makes edits easy to inspect. However, participant-reported correction frequency and failed-check proportions measure different outcomes and cannot be directly compared or attributed to a common cause.

#### **7.4 Limitations**

DAW100 covers selected operations under fixed model configurations and a common interface. Earlier trials informed task refinement, so the set is not an independently held-out benchmark. Automation has few checks, and some tools lack directly targeted tasks. Extending these areas and adding further starting projects would broaden the evaluation. State checks assess specified edits and preservation rather than musical quality.

The production study provides descriptive feedback from 14 experienced users in a guided session, without a controlled measure of efficiency gains. The original trust item combined visibility with reversibility despite the absence of project-wide undo, limiting interpretation of its responses. Removing the reversibility wording in the revised questionnaire does not change the basis of the existing results. The study does not establish benefits for beginners or people with specific input requirements. Model comparisons assess the combined model and interface; they cannot isolate the contribution of either component. Clearer object access, local note revision, resulting-state feedback, and broader undo are concrete directions for development whose effects remain to be evaluated.

### **8. Conclusion**

Sonara combines an implemented environment for language-directed music production, the DAW100 task set, and evaluation through editable project state. Across six model configurations and two runs per model, task success ranged from 35.0% to 83.0%. The strongest tested configuration completed most specified edits, while condition-level analysis exposed differences

in note execution and a shared difficulty preserving clip structure. The production study with 14 experienced users identified perceived benefits in exploration, intuitive control, and efficiency, alongside misinterpretation and recovery needs. Together, the studies show that reliable production assistance requires both achieving the requested change and preserving the surrounding work. Explicit object access, local editing, informative feedback, and reversible actions emerge as design priorities. The system and task set provide a concrete basis for examining these requirements while keeping musical decisions and subsequent revision available to the user.

## 9. References

- Cherry, E., & Latulipe, C. (2014). Quantifying the creativity support of digital tools through the Creativity Support Index. *ACM Transactions on Computer-Human Interaction*, 21(4), Article 21. <https://doi.org/10.1145/2617588>
- Copet, J., Kreuk, F., Gat, I., Remez, T., Kant, D., Synnaeve, G., Adi, Y., & Défossez, A. (2023). Simple and controllable music generation. *Advances in Neural Information Processing Systems*, 36, 47704–47720. <https://doi.org/10.52202/075280-2066>
- Elkins, A. C., Singh, S., Kieback, A., Blankenship, S., Amadasun, U. P., & Chadha, A. (2025). *DAWZY: A new addition to AI powered “human in the loop” music co-creation*. arXiv. <https://doi.org/10.48550/arXiv.2512.03289>
- Frid, E. (2019). Accessible digital musical instruments—A review of musical interfaces in inclusive music practice. *Multimodal Technologies and Interaction*, 3(3), Article 57. <https://doi.org/10.3390/mti3030057>
- Schick, T., Dwivedi-Yu, J., Dessi, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 68539–68551. <https://doi.org/10.52202/075280-2997>
- Wang, B., Li, Y., Lv, Z., Xia, H., Xu, Y., & Sodhi, R. (2024). *LAVE: LLM-powered agent assistance and language augmentation for video editing*. arXiv. <https://doi.org/10.48550/arXiv.2402.10294>
- Xie, T., Zhang, D., Chen, J., Li, X., Zhao, S., Cao, R., Hua, T. J., Cheng, Z., Shin, D., Lei, F., Liu, Y., Xu, Y., Zhou, S., Savarese, S., Xiong, C., Zhong, V., & Yu, T. (2024). *OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments*. arXiv. <https://doi.org/10.48550/arXiv.2404.07972>

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. *International Conference on Learning Representations*. <https://arxiv.org/abs/2210.03629>

Yu, D., Song, K., Lu, P., He, T., Tan, X., Ye, W., Zhang, S., & Bian, J. (2023). *MusicAgent: An AI agent for music understanding and generation with large language models*. arXiv. <https://doi.org/10.48550/arXiv.2310.11954>

2026 S.-T. Yau High School Science Award  
仅用于2026丘成桐中学科学奖论文公示

## **Appendix A**

### **Execution Settings and Scoring Conventions**

#### **A.1 Model and Context Settings**

The fixed snapshots are gpt-4.1-nano-2025-04-14, gpt-4o-mini-2024-07-18, gpt-4.1-mini-2025-04-14, gpt-4.1-2025-04-14, gpt-5-nano-2025-08-07, and gpt-5-mini-2025-08-07. All permit automatic tool selection and multiple calls per response, with a limit of 8,192 completion tokens per response. Reasoning tokens share this limit. GPT-5 configurations have no explicit temperature setting.

Top-level tool descriptions are limited to 1,024 characters and appended feedback to 2,000 characters per tool response. The API timeout is 120 seconds, with automatic SDK retries disabled. A response without tool calls ends execution. Otherwise, calls in round 16 are executed before scoring. Tool coverage requires a nonempty return without an error indication.

#### **A.2 Project Preparation and Shared Rules**

Before preparing each scene, the harness stops playback, removes non-master tracks and section markers, clears master effects, restores the default tempo, and checks for residual objects. The excluded tools are play, pause, stop, set\_position, show\_audio\_attachment, and set\_automation\_visible. The prompt directs models to get\_all\_tracks for track identifiers and status for clip identifiers.

The common conventions use 4/4 time, one-based bar and beat labels, exclusive section endpoints, and clip-relative note starts. Clip replacement is permitted for note editing unless identity retention is explicit. Multiple clips are allowed on initially clip-free tracks unless the task specifies one. Drum tasks use fixed symbolic note numbers; instrument and effect alternatives are specified in the prompt.

#### **A.3 Scoring Details and Research Records**

Timing tolerances are 0.02 beats for notes and clips, 0.02 bars for section boundaries, and 0.01 BPM for tempo. Normalised parameter targets allow approximately 0.01 absolute error. Note matching preserves multiplicity and accepts different ordering of equivalent note records. Direct

requirements are identified by comparing the final and initial task specifications and by explicit name, preset, copy, and identity requirements. Each task also has one joint MIDI-validity check covering clip membership, boundaries, and permitted note values.

The fixed collection version is audit2c-contracts-20260908. Task-order seeds are 20260909 and 20260910 for the two repetitions. Interrupted unscored trials resume from verified scenes; already scored trials are retained. The final 15 trials of the second repetition were collected after a connection-related pause, with the same frozen configuration. TSR intervals resample the 100 task IDs with replacement 4,000 times, retaining both executions, and use the 2.5th and 97.5th percentiles (seed 20260913).

Logs retain requests, tool arguments, full host responses, model-visible feedback, timestamps, scene checks, final states, scores, and token usage. Partial traces survive interruptions. Secrets are redacted. Source snapshots, dependencies, asset and host hashes, and execution order are archived with SHA-256 manifests after score replay and integrity checks. Infrastructure-failure attempts are excluded from scored interaction counts. Latency and attempted coverage are retained; no call-redundancy ratio is calculated without an equivalent-information minimum-call baseline.

## Appendix B

### Detailed Execution Results

**Table B1.** Condition Satisfaction and Task Success Across Both Runs

| Model        | All CSR (%) | Direct CSR (%) | TSR (%) | 95% CI for TSR |
|--------------|-------------|----------------|---------|----------------|
| GPT-4.1-nano | 90.0        | 50.3           | 35.0    | [26.5, 44.0]   |
| GPT-4o-mini  | 92.6        | 63.6           | 48.0    | [38.0, 57.5]   |
| GPT-4.1-mini | 95.1        | 86.7           | 62.5    | [53.0, 71.5]   |
| GPT-4.1      | 97.6        | 94.2           | 72.0    | [63.5, 80.5]   |
| GPT-5-nano   | 94.6        | 77.6           | 61.0    | [52.5, 69.5]   |
| GPT-5-mini   | 98.6        | 96.1           | 83.0    | [76.0, 89.0]   |

**Note.** Each model completed 200 trials. All CSR uses 2,442 checks per model: 330 direct requests, 1,912 preservation checks, and 200 validity checks. Direct CSR uses the direct requests. TSR requires every check to pass. CI = confidence interval; intervals use task-level bootstrap samples retaining both repetitions.

**Table B2.** Task Success and Interaction Counts by Task Group

| Model        | Level | Success          | Mean rounds | Mean calls | Limit reached |
|--------------|-------|------------------|-------------|------------|---------------|
| GPT-4.1-nano | L1    | 48/68<br>(70.6%) | 3.74        | 2.82       | 1             |
| GPT-4.1-nano | L2    | 16/76<br>(21.1%) | 6.07        | 5.39       | 1             |
| GPT-4.1-nano | L3    | 6/56<br>(10.7%)  | 6.86        | 6.68       | 1             |
| GPT-4o-mini  | L1    | 54/68<br>(79.4%) | 2.82        | 1.88       | 0             |
| GPT-4o-mini  | L2    | 24/76<br>(31.6%) | 4.74        | 3.91       | 4             |
| GPT-4o-mini  | L3    | 18/56<br>(32.1%) | 4.95        | 5.36       | 4             |
| GPT-4.1-mini | L1    | 62/68<br>(91.2%) | 2.74        | 1.78       | 0             |
| GPT-4.1-mini | L2    | 35/76<br>(46.1%) | 3.97        | 3.14       | 0             |
| GPT-4.1-mini | L3    | 28/56<br>(50.0%) | 5.05        | 4.89       | 2             |

| Model      | Level | Success          | Mean rounds | Mean calls | Limit reached |
|------------|-------|------------------|-------------|------------|---------------|
| GPT-4.1    | L1    | 66/68<br>(97.1%) | 2.62        | 1.68       | 0             |
| GPT-4.1    | L2    | 42/76<br>(55.3%) | 3.74        | 3.05       | 0             |
| GPT-4.1    | L3    | 36/56<br>(64.3%) | 4.62        | 4.59       | 0             |
| GPT-5-nano | L1    | 62/68<br>(91.2%) | 3.15        | 2.22       | 0             |
| GPT-5-nano | L2    | 36/76<br>(47.4%) | 6.63        | 5.95       | 9             |
| GPT-5-nano | L3    | 24/56<br>(42.9%) | 8.04        | 8.27       | 11            |
| GPT-5-mini | L1    | 64/68<br>(94.1%) | 3.16        | 2.21       | 0             |
| GPT-5-mini | L2    | 63/76<br>(82.9%) | 4.20        | 3.24       | 0             |
| GPT-5-mini | L3    | 39/56<br>(69.6%) | 6.25        | 5.54       | 1             |

*Note.* Two runs per model: 68 L1, 76 L2, and 56 L3 trials. Means include all scored trials and unsuccessful calls. Limit reached counts trials ending at the round cap.

**Table B3.** Audit of GPT-5-nano Trials Reaching the Round Limit

| Task / run | Recorded operation calls                   | Failed checks | GPT-5-mini      |
|------------|--------------------------------------------|---------------|-----------------|
| L2-02 R1   | Clips: 4 create, 4 delete; 7 note batches. | N             | Fail<br>3 calls |
| L2-03 R1   | Clips: 5 create, 2 delete; 5 note batches. | C, N          | Fail<br>4 calls |
| L2-07 R1   | Clips: 6 create, 3 delete; 6 note batches. | C, N, V       | Pass<br>3 calls |
| L2-10 R1   | Clips: 5 create, 3 delete; 4 note batches. | C, N          | Pass<br>3 calls |
| L3-03 R1   | Clips: 4 create, 0 delete; 4 note batches. | C             | Pass<br>7 calls |
| L3-04 R1   | Clips: 5 create, 0 delete; 4 note batches. | N, T          | Pass<br>5 calls |
| L3-14 R1   | Clips: 3 create, 3 delete; 5 note batches. | N, T          | Pass<br>3 calls |
| L3-17 R1   | Clips: 5 create, 4 delete; 5 note batches. | C, N, T, V    | Fail<br>3 calls |
| L3-18 R1   | Clips: 3 create, 1 delete; 9 note batches. | C, I, N       | Fail<br>6 calls |

| Task / run | Recorded operation calls                   | Failed checks | GPT-5-mini       |
|------------|--------------------------------------------|---------------|------------------|
| L3-26 R1   | Clips: 4 create, 4 delete; 3 note batches. | N, T          | Pass<br>4 calls  |
| L2-01 R2   | Clips: 5 create, 3 delete; 6 note batches. | C, N          | Fail<br>3 calls  |
| L2-03 R2   | Clips: 5 create, 4 delete; 9 note batches. | C, N          | Fail<br>3 calls  |
| L2-04 R2   | Clips: 5 create, 3 delete; 4 note batches. | C, N          | Fail<br>3 calls  |
| L2-10 R2   | Clips: 4 create, 3 delete; 4 note batches. | C, N, T       | Pass<br>3 calls  |
| L2-27 R2   | Clips: 7 create, 0 delete; 0 note batches. | S, T          | Pass<br>2 calls  |
| L3-02 R2   | Clips: 5 create, 3 delete; 4 note batches. | N, S          | Fail<br>12 calls |
| L3-10 R2   | Effects: 8 add, 2 remove, 1 clear.         | E             | Pass<br>5 calls  |
| L3-17 R2   | Clips: 5 create, 4 delete; 5 note batches. | C, N, T       | Fail<br>3 calls  |
| L3-18 R2   | Clips: 4 create, 3 delete; 4 note batches. | N             | Fail<br>9 calls  |
| L3-26 R2   | Clips: 4 create, 3 delete; 4 note batches. | C, N          | Pass<br>4 calls  |

**Note.** All twenty GPT-5-nano trials failed at the round cap. R1/R2 identifies the repetition. Clip and note-batch counts are requested calls, including any unsuccessful calls. N = note pattern; C = clip structure; T = track identity/count; V = MIDI validity; S = sections; I = instrument; E = effects. Categories overlap. The last column reports GPT-5-mini on the same task and repetition, not additional trials.

## Acknowledgement

The original inspiration for this research came from my own campus life. In my first year of high school, I joined the flag football team and the soccer club, hoping to create original team songs for them. I have studied piano since childhood and have always loved music composition. In 2021, I participated in the call for songs for the Beijing Winter Olympics. Although my original work, Spirit of Youth at the Winter Olympics, was not selected, that experience planted the seed of music creation in me. Traditional music production is complicated and time-consuming, making it difficult to balance academic work with the creative needs of school clubs. After I began learning AI programming, I experimented with Suno AI for music creation, but the tool only supports end-to-end music generation and does not allow section-level editing. If I wanted to adjust the bass volume or modify a chord, I had to regenerate the entire piece. This made me realize that what I needed was not an AI that would compose music directly for me, but rather an AI tool that could assist people in operating music production software. From this core idea, Sonara was born.

Once I began implementing the project, I quickly encountered the central challenge: how could AI translate a creative instruction into the correct operation within music production software? Drawing on the design philosophy of the Model Context Protocol (MCP), I built a tool-layer architecture that encapsulated operations involving tracks, notes, timbres, and effects into callable interfaces, while also providing project-state queries so that the AI could choose subsequent actions based on the actual state of the project. When the first note successfully appeared in the piano roll through a voice command, the entire system began to function for the first time. It was also the first time I truly saw that natural language could serve as a bridge between a creative idea and a specific editing operation.

This project spans multiple fields, including music theory and computer science, and throughout the research process I engaged in extensive self-study and sought guidance from others. First, I would like to thank my homeroom teacher, Ms. Li Jinghong of Beijing No. 80 High School. From the initial selection and establishment of the research topic to the repeated revision and refinement of the paper, Ms. Li provided me with unpaid guidance, full trust, and ample space for independent exploration. She reminded me that the purpose of technology is not to replace human creativity, but to amplify the creative power of people. This statement

continued to remind me, especially during the countless times I debugged the system and nearly fell into the trap of focusing only on technology: what I was building was not a cold technical tool, but a medium that could help users gain greater creative freedom. The successful completion of this project would not have been possible without Ms. Li's continued support and understanding.

I would also like to express my special gratitude to Ms. Wang Heng. Ms. Wang completed undergraduate studies in both chemistry and computer science, has a research background in quantum computational chemistry, and serves as the faculty advisor of the Kaiyang AI Study Club. As one of the core student founders and the first president of the club, I brought together a group of students who shared a strong curiosity about artificial intelligence to explore AI-related research topics. It was during discussions within the club that I first proposed this idea, which at the time seemed somewhat unrealistic. Rather than dismissing it, Ms. Wang carefully evaluated the feasibility of the project and helped point me toward a research direction. During nearly a year of research, she provided selfless and valuable guidance in many areas, including theoretical thinking, data analysis, and engineering implementation. We discussed and refined the design of the tool interfaces together and repeatedly examined the connections among voice interruption, project-state queries, and editing operations. More importantly, she helped me truly realize that a high school student can begin from nothing and explore a solution that does not yet exist in the market, as long as the reasoning is rigorous, the work is grounded in reality, and the student has genuine enthusiasm for research. This selfless and unpaid support, offered without any expectation of personal gain, became one of the strongest sources of support throughout my project.

Real-world usability evaluation was an important part of the workload of this project. To test Sonara's performance in actual creative workflows, I conducted two studies: an automated task evaluation and a user study and was fortunate to invite experienced music producer Mr. Liu Mo to participate as a volunteer tester. From the perspective of an industry professional, he provided many demanding yet valuable comments. Many product details, ranging from precise command control to fault tolerance for contradictory instructions, were influenced by this valuable feedback. I am especially grateful that he evaluated my project from the perspective of an equal professional peer. The core code, system implementation, all experimental data, and paper writing for this project were completed independently by me. The assistance provided by

all of the teachers mentioned above fell within the scope of reasonable, unpaid guidance, and all external assistance has been fully disclosed.

Finally, I would like to thank my parents for providing the experimental equipment, covering the cost of large language model API usage, and, even more importantly, staying with me through countless late nights spent debugging code, running experiments, and reading research papers.

During this study, from January 2026 to September 2026, I used ChatGPT, including the GPT-5.5, GPT-5.6 Sol, and GPT-5.6 Terra models, for an average of approximately five hours per week. The AI tools were primarily used to assist with organizing DAW100 video transcript materials and summarizing themes, providing suggestions regarding the Sonara framework, troubleshooting program and compilation errors, and improving MCP interface validation and exception handling. They were also used to assist in organizing the results, statistical metrics, failure records, and questionnaire feedback from the two experiments, as well as for English grammar checking, terminology consistency, readability improvement, and document formatting.

The core work of this research, including the formulation of the research question, system and methodology design, experimental design and implementation, data collection, technical implementation, interpretation of results, research conclusions, and the professional judgments presented in the paper, was completed independently and manually by myself. All AI-generated or AI-provided suggestions, organized materials, code-related recommendations, and textual revisions were manually reviewed, evaluated, and, when necessary, modified by myself before any decision was made regarding their final adoption.

Looking back at the original intention of the project, Sonara is designed to let AI take over repetitive and mechanical operations while leaving creative decision-making in the hands of the user. Users do not need to be familiar with complex operations such as loading plug-ins. With a natural-language instruction such as “Give me a warm-sounding piano passage,” they can begin creating music, lowering the barrier to entry and allowing the focus to return to creativity itself. Sonara still has many areas that require further iteration and improvement, and I will continue to refine it in the hope that, in the future, it can help more people enter the world of music creation with fewer technical barriers.