RIME: ENABLING LARGE-SCALE AGENTIC MUSIC POST-PRODUCTION

Anonymous authors
Paper under double-blind review

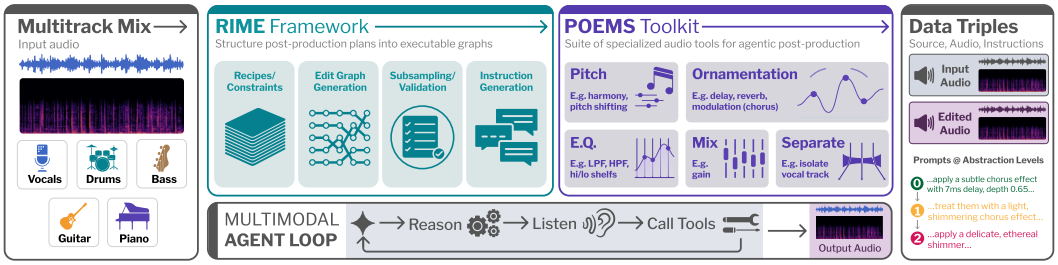


Figure 1: Agentic music post-production with RIME. Starting from a multitrack mix, RIME creates rule-based plans and calls POEMS tools to generate data triples of (`source audio`, `output audio`, `editing instruction`). Downstream multimodal agents then use POEMS to implement an agentic loop in which they realize the provided instruction.

ABSTRACT

Almost every piece of recorded music you have ever heard was modified before it reached you; commercial releases rarely spring fully formed from a musician's mind. Despite the promise of music generation models for one-shot output, such fine-grained *iterative refinement* workflows are a complementary problem, and largely out of their reach. There is also a gap for musicians: while they can express what they want to hear, not all have the facility with studio production tools to implement the complex set of actions needed to realize these intuitions. We formalize this task as *agentic post-production*, wherein individual aspects of a track are targeted, refined, and combined into a final version. We argue the bottleneck is data: existing corpora do not reflect how realistic post-production chains map onto the vocabulary musicians and engineers actually use. We observe that there is a language for modifying recorded music that is dense, consistent, and learnable, and to leverage this we introduce the **R**ule-based **I**nstructions for **M**usic **E**diting (**RIME**) framework, which generates realistic paired edit-instruction data from any baseline music dataset grounded in canonical methods, design patterns, and constraints derived from real production workflows. RIME leverages POEMS, a new toolkit that combines stem separation, mixing, and common studio effects for use by multimodal agents. We use POEMS and RIME to generate 15,000 pairs of edit instructions and ground-truth audio, and use this data to evaluate existing multimodal LLMs as agents on this task, showing persistent challenges in current models' post-production capabilities. We also demonstrate RIME's ability to improve post-production agent performance via supervised fine-tuning using synthetic data generation. We see RIME as an early step toward iterative musical agents, collaborative systems that could transform music production much as interactive coding agents have reshaped software engineering. We include all code for RIME at: `https://anonymous.4open.science/r/RIME-0116/`

1 INTRODUCTION

There is a pervasive loop in the studio. When a producer listens to a recording they're working on, they will often hear something wrong, or something that could be better. They may isolate the offending stem, apply a chain of effects, blend it back in, and listen again. The musician being

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

recorded may hear something else and make a request. The producer implements a fix, then listens again. Then again. Then again.

This loop has its own peculiar language. When a musician asks for something "warmer" or "bigger," a seasoned engineer can translate it into a specific recipe (e.g. a high-shelf cut at 8 kHz, parallel compression on the drum bus, and a tempo-synced reverb send on the vocal) and run it. The translation between what music *feels* like and the sequence of operations that produces that feeling is not, in fact, magic. It is a dense and surprisingly consistent vocabulary, built up over decades of studio practice.

Today's AI music systems are typically not part of this loop. Yet, we have good reason to desire them to be: it admits an iterative, interactive workflow that is musician-centered at its core. It is now reasonable to ask whether a software agent could join it, because these agents live in loops of their own: perception, action, reasoning, and tool use. Indeed, this would also be a strong testbed for such agents. Each iteration in this loop couples a perceptual judgment, an instruction context, a large assortment of potential tools, planning, and reasoning. What would it take, then, to bring an agent's loop into dialog with this *post-production* loop?

We argue that what is needed is a general framework for modifying music, beyond the capabilities of one-shot generators. To this end, we introduce the **R**ule-based **I**nstructions for **M**usic **E**diting (**RIME**) framework, which aims to generate large-scale music editing data that follows a process closer to what you would see in studio music production. RIME starts from individual mixes, creates edit "recipes" using a large vocabulary of common studio engineering rules, constraints, and design patterns, and executes a series of tool calls to isolate sources, apply edits, and create a final mix. This then creates supervision for agents to do the same. Additionally, RIME creates instructions at varying levels of abstraction to describe the edit chain for a given recipe, mimicking the variation in how precisely these intents are described in the real world. To create editing data, RIME leverages a newly built suite of tools that allow for **p**itch, **o**rnamentation (various effects), **e**qualization, **m**ixing, and source **s**eparation (POEMS) edits, which we call POEMS.

RIME is also unique in that it can be extended to any dataset containing musical tracks, and especially designed for *mixed* tracks (or even *generated* ones, potentially, which normally come without individual instrument stems). We demonstrate using RIME in conjunction with state-of-the-art music understanding models to generate triplet edit data from untagged musical mixes.

Overall, our contributions are:

1. We formalize **agentic post-production** as an iterative task: starting from a mixed track and a production intent (an arbitrarily clear instruction), an agent targets individual elements, refines them, and recombines them into a finished mix
2. We introduce RIME, **a scalable, reusable data generation framework** for transforming *virtually any corpus of (contemporary Western) musical audio* into a structured set of (input, output, instruction) triples. RIME encodes recipes, motifs, ordering constraints, and pattern policies of studio post-production, in part calibrated from datasets of real production chains, and produces instructions at multiple levels of abstraction
3. To support this framework, we introduce POEMS, **an audio post-production toolkit** spanning over 20 operators exposed via the Model Context Protocol for use by tool-calling agents
4. We run **a paired triplet benchmark** with ≈15,000 (input, output, instruction) triples generated with RIME on MusicCaps Agostinelli et al. (2023), enabling evaluation across instrument families, edit categories, and abstraction levels
5. We conduct a **listener study** to validate that musically experienced users can reliably match edits and their corresponding instructions across abstraction levels
6. We develop POEMS Agent Framework to design and evaluate **zero-shot agent** baseline multi-step tool-calling audio-language model agents, surfacing major limitations to current-generation models' abilities on this task, such as their ability to assemble the correct chain of operators vs. parameterize those operators correctly
7. We **fine-tune** an open-weight LLM orchestrator with RIME-synthesized data and demonstrate improvements over the zero-shot agent baselines

2 Related Work

Many previous text-prompted generative editing tasks have relied solely on paired instruction/audio data and audio-language embedding improvements to perform edits (Liang et al.; Tsai et al., 2024; Chu et al., 2025; Zhang et al., 2024; Manor & Michaeli, 2024; Yang et al., 2026; Lan et al., 2026). While this allows for certain edits without the need for supervised (input, output, text) data, the lack of ground-truth data makes these methods difficult to evaluate systematically. Several general audio editing frameworks that leverage triplet data (input audio, edited audio, instruction); however, their datasets largely focus on more general edits such as addition, deletion, replacement (via text-to-audio synthesis), inpainting, and superresolution (Wang et al., 2023; Liang et al., 2025; Jia et al., 2025; Ellis et al., 2025). InstructME (Han et al., 2024) creates paired triplet edit data from musical variations, focusing on only a small subset of production edits such as remixing, adding, replacing, extracting, and removing.

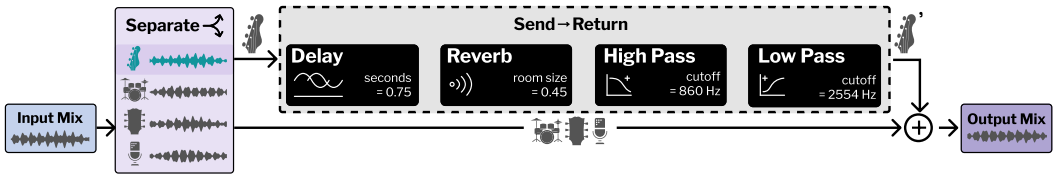
Agentic editing is a relatively unexplored area. SonicMaster (Melechovsky et al., 2025) uses an agent to repair five categories of degradations (EQ, Dynamics, Reverb, Amplitude, and Stereo) from audio, but does so using DDIM inversion and noise injection. LLM2Fx-Tools (Doh et al., 2026) creates a benchmark from MedleyDB (Bittner et al., 2014; 2016), randomly sampling musically realistic FX chains on single sources and uses a language model to create specific natural language instructions. It then fine-tunes a (text-only) LLM tool-calling agent to derive Fx chains from instructions. Although this is the closest work to a post-production agent, it operates only on single stems, does not rely on an iterative listening workflow, and does not differentiate the multiple levels at which an edit may be described in natural language.

Several existing frameworks have analyzed the mapping between abstract natural language and studio effects. SocialFx (Zheng et al., 2016) analyzed common abstract descriptors of three individual effects. Further work analyzed effect chain recommendation from natural language descriptors (Stasis et al., 2017). Recently, MixAssist (Clemens & Marasovic, 2025) introduced a dataset capturing conversations between amateur and professional music producers during a mixing session. There is no existing work that has leveraged these findings to motivate actions taken by a tool-calling agent, nor have these language/fx mappings been extended to the full context of music post-production.

In all, no existing benchmarks match the characteristics of real-world post-production loops described earlier. For this, such a benchmark would need to (i) apply edits to individual stems and introduce them back into a mix, (ii) be able to handle instructions at varying abstraction levels, and (iii) support an iterative, listen-and-tweak workflow. Here, we introduce exactly this suite of components.

3 RIME: Rule-based Instructions for Music Editing

We assume that a user begins with an input audio clip x with associated/extracted metadata $m(x) = (\mathcal{S}_x, g_x, t_x, \kappa_x)$ (set of available stems, genre tags, tempo, key). We seek to simulate many edit workflows to produce a dataset, outlined in Figure 1 and explained below.



Abstraction Level 0 (AL0)

"... add a delay of 0.75 seconds, then reverb with room size 0.45, then highpass at 862 Hz, then lowpass at 2554 Hz...send level 0.51 and return level 0.72..."

Abstraction Level 1 (AL1)

"...add a long, trailing, spacious echo...then a dark, damped, muffled, gloomy reverb...wide, full-width...then remove low mids... then remove upper mids"

Abstraction Level 2 (AL2)

"Give the bass a noticeable, tasteful, present spacey depth with a muffled, distant wash that sits under the dry signal, supportive and band-limited."

Figure 2: A RIME edit graph instantiated from the `target_space_send_synced` recipe on a `bass` stem. From this graph, we generate instructions at three distinct levels of abstraction.

3.1 THE POEMS TOOLKIT

First, we need a suite of tools to enact a given edit workflow. Modern LLM agents can sometimes write code to create such tools, but we see no need to reinvent the wheel at runtime. Producers often have favored audio plug-ins, and integrating with these similarly requires the agent to use existing tools. At the same time, we need a stable, canonical, and agent-oriented set of tools for a robust evaluation. For this purpose, we introduce the Pitch, Ornamentation, Equalization, Mixing, and Separation (POEMS) toolkit, which defines an operator catalog $\mathcal{O}$ for end-to-end music post-production. Each operator $o \in \mathcal{O}$ has an associated parameter space Θ_o.

To edit an individual source within a mixed track, we must first isolate it using source separation. We use the 6-stem variant of Demucs (Rouard et al., 2023) to separate $x = x_s + r_s$, where x_s is the target stem and $r_s = x - x_s$ is the residual. This restricts editable stems to vocals, drums, bass, piano, and guitar (we attempted to use SAM Audio (Shi et al., 2025), but found it produced poor results in this setting). After processing, we mix the edited stem back with its residual $\tilde{x} = \tilde{x}_s + r_s$.

We provide two pitch edits for vocal stems: pitch correction (autotuning) and harmony generation. For both edits, we use estimated f_0 contour from CREPE (Kim et al., 2018), S-KEY key detection (Kong et al., 2025), quantization, and resynthesis with PSOLA (Valbret et al., 1992) to preserve formant structure. We implement four ornamentation and equalization effect families with Pedalboard (Sobot, 2021): time-based (delay, algorithmic reverb), modulation (chorus, phaser), dynamics (compressor, peak limiter, waveshaping distortion, noise gate, split-band de-esser), and equalization (high-pass, low-pass, high-shelf, low-shelf, and parametric peaking filters). In all, this gives us a substantial toolkit capable of enacting a wide variety of desired edits (full list Appendix A).

3.2 RIME COMPONENTS

Recipes, Graphs, and Motifs The central abstraction RIME relies on is an *edit graph* $G = (V, E)$, a directed acyclic graph whose nodes $v \in V = (o_v, \theta_v, s_v)$ each apply an audio operator $o_v \in \mathcal{O}$ with parameters $\theta_v \in \Theta_{o_v}$ to audio. This is similar to an FX chain in common audio production parlance, except that it allows parallel branches (via audio sends which copy audio off from the stream, and audio returns which re-mix modified versions of those copies into the original stream).

Hand-specifying enough graphs (e.g. thousands) to form a robust dataset would be infeasible. Rather, we need a way to *generate* such graphs at scale. Fortunately, audio production follows many *recipes*, defined here as symbolic templates for edit graphs with particular aesthetic goals (see example in Figure 2). Each recipe $r \in \mathcal{R}$ is specified using a schema $r = (B_r, \varphi_r, \lambda_r, H_r)$:

- `bindings` B_r: (references to clip metadata resolved at plan-generation time, e.g. *key*)
- `when` φ_r (eligibility predicate, e.g. `vocal_harmony_support` requires *vocal stem* and *key*)
- `weight_rules` λ_r (multipliers to indicate how common certain recipes should be)
- `sequence` H_r (outline for applying the recipe, normally `SEPARATE` $\rightarrow$ `PROCESS` $\rightarrow$ `MIX`)

`PROCESS` contains the recipe's actual processing sub-graph. `SEPARATE` and `MIX` scaffold it with stem isolation and residual recombination steps, so every recipe operates in a stem-aware fashion.

To share structure *across* recipes, we factor commonly co-occurring sub-chains into reusable units we call *motifs*. A motif is a named, typed sub-graph that can be expanded inline within any recipe that references it. For example, the `retro_tilt_eq` motif expands to `high_shelf()` $\rightarrow$ `lowpass()`, capturing the joint high-frequency roll-off characteristic of a vintage sound.

Parameter Priors To generate plausible-sounding audio effects, we need to sample their parameters in such a way that we favor commonly used settings while allowing naturalistic variation across edits. We take an empirical approach to this by estimating prior sampling distributions from observed processor settings, which we call *generation priors*. Each prior π_o is a distribution over operator o's argument space Θ_o. Recipes specify which prior π_o to use, so the same operator can use different priors in different recipes (e.g. to account for stylistic variation).

We collect processor settings from MixParams (Clemens & Marasovic, 2025) and SocialFX (Zheng et al., 2016), using MixParams for equalization and compression, including separate vocal-compression priors, and SocialFX to supplement general compression thresholds and ratios. To

balance coverage across sources and settings, we give contributing datasets equal weight, songs equal weight within MixParams, and distinct parameter values equal weight within SocialFX. We exclude controls with unspecified/unclear units (e.g. attack and release), and choose heuristic priors for these.

For each parameter j, we set heuristic bounds $\mathcal{D}_j = [a_j, b_j]$ to exclude settings we expect to produce excessively weak or extreme effects, and fit priors using observations within that domain. We then fit on $z = h_j(\theta_j) \in [0, 1]$, applying a log transform before normalization for log-scaled parameters. Candidate distributions $f_m(z; \eta_m)$ include uniform, Gaussian, beta, and two-component Gaussian mixture models, plus power laws in original units for log-scaled parameters. We bound-constrain all candidates and fit by weighted maximum likelihood:

$$\widehat{\eta}_m = \arg\max_{\eta_m} \sum_i w_i \log f_m(z_i; \eta_m). \tag{1}$$

Then, we select:

$$m^\star = \arg\min_m \left[-2N_{\text{eff}} \sum_i w_i \log f_m(z_i; \widehat{\eta}_m) + d_m \log N_{\text{eff}} \right], \tag{2}$$

where d_m = number of fitted parameters and $N_{\text{eff}} = 1/\sum_g W_g^2$, with W_g denoting total weight of one song or settings group. This is a BIC-like approximation to balance fit against complexity. We only include contributing sources with ≥ 30 observations and at least three songs or distinct parameter settings groups to avoid overfitting to idiosyncratic examples. We use a custom web-based UI to have an audio engineer double-check parameter distributions by ear, described in Appendix F.

Constraints Plausible parameter values do not guarantee useful edits: effect order and recipe applicability also matter. RIME applies two classes of constraints when sampling edit graphs.

Chain Order Constraints. We use the default order `EQ` $\prec$ `Dynamics` $\prec$ `Distortion` $\prec$ `Modulation` $\prec$ `Time-Based Effects` $\prec$ `Leveling` (see Appendix B for specific references regarding effects ordering). Individual chains can specify a different order when needed; for example, ambiance recipes apply delay and reverb *before* filtering the wet signal, while vocal harmony applies pitch shifting, gain adjustment, then filtering.

Pattern Policy Constraints. We use policy weights to favor edits more appropriate for each clip and target instrument, such as reducing the weight of modulation on drums (very unusual) and increasing the weight of vocal compression (very common). Given clip metadata m and slot assignment b, we combine the recipe-local weight $\lambda_r(m, b)$ with applicable policy multipliers $\lambda_q(m, b)$:

$$\omega = \lambda_r(m, b) \prod_q \lambda_q(m, b). \tag{3}$$

An audio engineer with professional production credits reviewed, auditioned, and refined these rules alongside the recipes, motifs, and priors using our web-based UI (Appendix F). We provide the full policy in Appendix B.3 and component details in Appendix B.

3.3 Instruction Generation

We combine the recipes, motifs, priors, and constraints above to construct a candidate pool $\mathcal{P}$ (Algorithm 1). For each clip x, we resolve recipe bindings against $m(x)$ and generate J parameter draws per eligible assignment. We retain each graph with weight ω for subsequent subsampling.

In post-production, there are varying degrees of granularity with which you can describe the same set of edits. A sound engineer may be able to describe an edit with extreme specificity (e.g. "apply a chorus with 20 ms delay and .65 depth"), while a musician desiring a similar result may say e.g. "apply a delicate shimmer." We account for this by generating instructions at three *abstraction levels*:

AL0 Describes the edit graph directly. All effect names, exact parameter values, and operator order are preserved in the generated instruction (e.g. language of professional mix engineers).

Algorithm 1 Recipe-based plan pool construction.

Inputs: annotated clips $\mathcal{D}$, recipes $\mathcal{R}$, motifs, parameter priors $\{\pi_o\}$, and J draws per assignment.
Output: a list $\mathcal{P}$ of candidate graphs and selection weights. Initialize $\mathcal{P} \leftarrow [\,]$.

For each clip $x \in \mathcal{D}$

↪ 1 **Read clip annotations.** Obtain $m(x)$: available stem descriptions, genre, tempo, and key.

For each recipe $r \in \mathcal{R}$

↪ 2 **Assign recipe slots.** Generate each allowed assignment b, such as a choice of target stem. Apply the following steps to each assignment.

↪ 3 **Check recipe requirements.** Skip the assignment if $\varphi_r(m, b)$ is false.

↪ 4 **Apply selection rules.** Compute the weight ω from recipe and global rules. Skip candidates that a rule rejects or gives $\omega \leq 0$.

Repeat steps 5–8 for J ***parameter draws per surviving assignment***

↪ 5 **Expand motifs.** Fill recipe template H_r with assignment b and replace motifs with constituent steps.

↪ 6 **Resolve arguments.** Use fixed values, clip annotations, and recipe's priors to get G.

↪ 7 **Validate the graph.** Check arguments and constraints (§3.2); stop with error if either check fails.

↪ 8 **Add to candidate pool.** Append $(x, r, b, \text{SERIALIZE}(G), \omega)$ to $\mathcal{P}$ for subsequent work.

AL1 Summarizes output of AL0. All effect names are preserved, but exact parameters and operator order are not. The distribution of parameter values is given as context to abstractly describe parameterization (e.g. reverb with a room size of .2 might be described as "subtle" while .9 might be described as "vast").

AL2 Summarizes the output of AL1, focusing on overall character and magnitude of an edit chain. Does not directly specify effect names, parameter values, or parameter order.

All instructions are generated using NVIDIA Nemotron 3 Super 120B (Blakeman et al., 2025) as a held-out model; more details in Appendix C. We also provide a UI for auditing these instructions against graphs, shown in Appendix F.

3.4 Data Validation

As a final check, we seek to ensure instructions and recipes are *audible*, *stylistically appropriate*, and *accurate* at every abstraction level. We therefore screen examples for four types of errors: **(1) abstraction-level violations:** instructions fail to follow the requirements for their abstraction level; **(2) inconsistencies:** instructions misinterpret the preceding level's text; **(3) stylistic inconsistencies:** instructions describe edits that conflict with the source material, using MusicCaps (Agostinelli et al., 2023) captions as context, and **(4) parameter incongruity:** edits combine parameter values in ways that produce implausible processing. We use Qwen3 235B (Yang et al., 2025) as an LLM judge and discard examples it flags under any criterion. We then rank the remaining examples by cosine similarity between source and edited audio using MERT (Yizhi et al., 2023) embeddings; we remove the lowest-scoring 5% to screen out extremely atypical edits, then keep the next $5,000$ by audibility as our primary set. In Appendix D, we provide full validation procedure details.

4 Experiments

POEMS-AF: the Agent Framework We introduce a multi-turn tool-calling framework for an agent to use POEMS to isolate a target stem, process it, and mix it back into the original track. The agent first listens to the input and proposes a tool-use plan based on the user's instruction. At each step, it listens to the current audio and may revise its plan before selecting the next tool or returning the result. For malformed tool calls, we allow n recovery attempts before invoking the tool with default arguments (Appendix A); we set $n = 1$. We allow at most 20 tool-call steps (excluding recovery attempts). If the agent does not return edited audio within this limit, we declare **agent failure** and use the unedited audio as evaluation outputs.

Generating Synthetic Data with RIME We use RIME to synthesize evaluation triplets (`input_audio, edited_audio, instruction`) starting from a corpus of full mixes with no available stem-level annotations. We use Music Flamingo (Ghosh et al., 2026) to identify instruments in MusicCaps (Agostinelli et al., 2023) tracks, and keep only tracks that contain at least 2 stems and have at least one stem in $[\text{vocals}, \text{bass}, \text{drums}, \text{guitar}]$, resulting in 2899 tracks. From these, we generate $79,905$ plans and subsample $25,000$ as described in **??**. Using the LLM filtering described in Section 3.4, we obtain $15,335$ valid clips and then further subsample the top $5,000$ using the MERT ranking described in the same section.

For SFT training data, we sample $3,000$ random instrument combinations, with priors derived from (Nunes & Ordanini, 2014; Pereira et al., 2023), generate $105,741$ plans, subsample to $25,000$, and then filter using the LLM judge, resulting in $20,098$ plans. We then use coverage-constrained stratified random sampling, where we select $1,000$ of these plans such that all recipes used in creating ground-truth edit data are represented in the training set. For each chosen plan, we generate instructions at all three abstraction levels, resulting in $15,000$ unique evaluation and $3,000$ unique SFT training (`source_audio, output_audio, editing_instruction`) triples.

Supervised Finetuning We perform supervised fine-tuning on an agent using GEMMA 3N (Team et al., 2024) orchestration. We fine-tune *only* the reasoning step and do not enforce any specific tool calls. We use low-rank adaptation (Hu et al., 2022) with a rank of 16, α of 16 and learning rate of 2×10^{-4}. We train for one epoch on the 3,000 data triplets from the training dataset. We keep the multi-turn tool-calling framework identical to the zero-shot agent for comparison.

Evaluation Metrics We use two primary modes of evaluation: audio similarity and edit-graph similarity. We use Fréchet Audio Distance (Roblek et al., 2019) and Kernel Audio Distance (Chung et al., 2025) in the MERT (Yizhi et al., 2023) embedding space. We also introduce edit similarity (Δsim_a), which measures the similarity between the direction of change (in the MERT embedding space) of the agent's output vs. input and the direction of change of the ground-truth edited audio from RIME. For graphs, our primary metric is Graph F1, which is the arithmetic mean of four F1 scores on graph *kind*, *operator*, *stem*, and *edge*. In Appendix G.1, we provide several additional evaluations to corroborate the primary metric results.

Human Evaluation To test whether instructions adequately describe audible edits across abstraction levels, we conduct a listening study using 30 *original–edited audio* pairs and rate three instructions per pair: the matching instruction, a hard negative targeting the same instrument and often similar effects, and another distractor, rating the correspondence of each with the audible edit on a 1–5 scale.

5 RESULTS

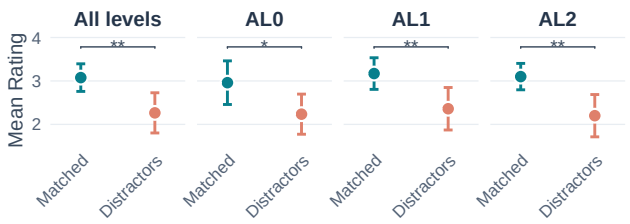


Figure 3: Estimated marginal audio-correspondence ratings on a 1–5 scale for true edit instructions vs. distractors, adjusting for item differences. Listeners rated matching instructions above distractors across abstraction levels.

Human listeners can identify edits across abstraction levels We recruited 10 participants who all had music and audio backgrounds (via Prolific filters). Of the 10, 3 identified themselves as "Mix engineer," 3 as "Music producer," and 4 as "Musician or performer." 3 had 10–19 years of experience, 4 had 5–9 years, 2 had 2–4 years, and 1 had less than 2 years. They described their listening setup as headphones (6), headphones and studio monitors (3), and studio monitors only (1). We collected 900 ratings across original-edited audio pairs (10 participants $\times$ 30 items $\times$ 3 ratings = 900 observations).

In Figure 3, we plot estimated marginal means from a regression predicting rating as a function of description role (the true matched description vs. distractor) interacted with abstraction level with source-audio (item) fixed effects and participant-clustered standard errors. We find that human listeners rated the true edit description as **significantly more matched to the audio** across all

Table 1: Evaluation using pooled MERT clip embeddings. Graph F1 averages node-kind, stem, operator, edge F1. ↑ or ↓ indicate which direction is better.

			Metrics				
	Category	N	FAD ↓	FAD_∞ ↓	KAD ↓	Δsim_a ↑	Graph F1 ↑
GPT-Audio-Mini	*Overall*	15000	0.025	0.022	0.016	0.637	0.821
	AL0	5000	0.030	0.025	0.024	0.718	0.869
	AL1	5000	0.039	0.034	0.014	0.675	0.846
	AL2	5000	0.046	0.039	0.029	0.519	0.747
Gemini 3 Flash	*Overall*	15000	0.041	0.038	0.024	0.528	0.634
	AL0	5000	0.066	0.061	0.070	0.550	0.624
	AL1	5000	0.058	0.056	0.020	0.561	0.670
	AL2	5000	0.045	0.039	0.027	0.473	0.608
Gemma 3n	*Overall*	15000	0.085	0.083	0.100	0.609	0.857
	AL0	5000	0.082	0.083	0.109	0.687	0.906
	AL1	5000	0.139	0.147	0.127	0.611	0.882
	AL2	5000	0.096	0.095	0.096	0.528	0.782
Gemma 3n SFT	*Overall*	15000	0.033	0.029	0.022	0.615	0.858
	AL0	5000	0.035	0.030	0.030	0.745	0.936
	AL1	5000	0.069	0.059	0.047	0.633	0.895
	AL2	5000	0.064	0.056	0.077	0.465	0.743

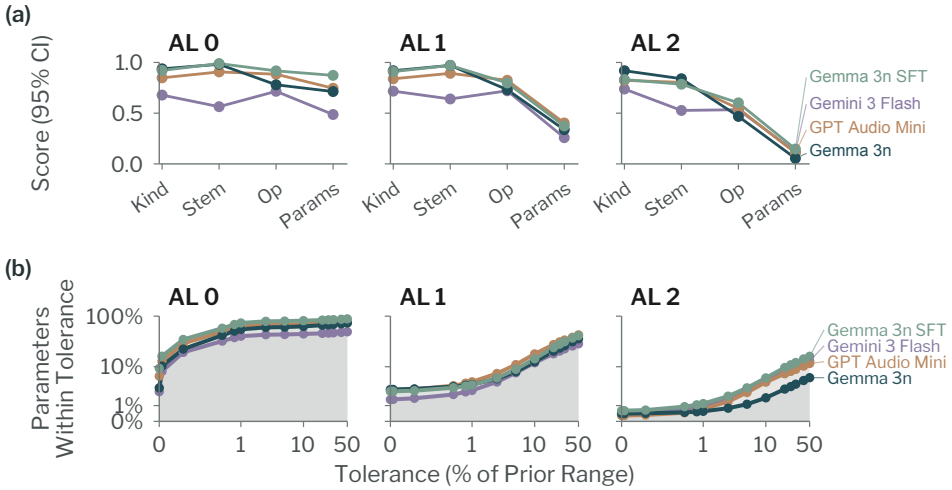


Figure 4: Graph and parameter recovery across abstraction levels. (a) Kinds, stems, and operators use F1; Params uses 1−normalized MAE (b) Parameter accuracy as numeric tolerance increases. Error bars show 95% CIs.

abstraction levels ($p = 0.002$ for AL1, AL2, and $p = 0.02$ for AL0). The mean rating is higher for the true description compared with both distractors at all levels of abstraction (all $p < 0.04$ after Benjamini-Hochberg correction across 6 tests, except AL0 Distractor 1 $p = 0.12$). Additionally, the mean rating was higher for matched than for both distractors for all 30 items. Together, this provides strong evidence that the summarization and abstraction pipelines preserve necessary information pertaining to specific edits.

Post-Production agents struggle in zero-shot settings Across 15,000 attempts per model (5,000 target edits × 3 abstraction levels), GPT-Audio-Mini produces 13,636 successful outputs (90.9%), Gemini 3 Flash 13,563 (90.4%), and Gemma 3n 14,880 (99.2%). Most zero-shot failures (95.9%) reach the step limit without completing the edit, others include invalid responses. We treat failed runs as no-ops with the input audio as output and an empty predicted edit graph.

Results are shown in Table 1. In summary: GPT-AUDIO-MINI has the highest target-similarity increase (0.637 vs. 0.609 for GEMMA 3N and 0.528 for GEMINI 3 FLASH), and the best FAD/FAD$_\infty$/KAD. GEMMA 3N wins on Graph F1 (0.857 vs. 0.821 for GPT-AUDIO-MINI and 0.634 for GEMINI 3 FLASH), but also has worst FAD (0.085). GEMMA 3N completes more requests and recovers more graph structure than GEMINI 3 FLASH, yet produces a less similar output *distribution*.

Abstraction challenges parameter recovery Edit similarity reduces as instructions move from explicit parameter values (AL0) toward evocative descriptions (AL2). GPT-AUDIO-MINI falls from 0.718 → 0.519, and GEMMA 3N from 0.687 → 0.528. GEMINI 3 FLASH improves slightly from AL0 to AL1 (0.550 → 0.561), then falls to 0.473 at AL2. GPT-AUDIO-MINI leads among zero-shot agents at AL0 and AL1. GEMMA 3N is slightly ahead at AL2.

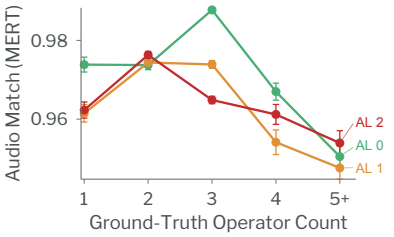


Figure 5: Avg. MERT output–target similarity by ground-truth operator count. Error bars show 95% CIs.

Parameter recovery is the clearest decline (Figure 4a): 0.746 → 0.403 → 0.109 for GPT-AUDIO-MINI, 0.713 → 0.337 → 0.055 for GEMMA 3N, and 0.487 → 0.258 → 0.118 for GEMINI 3 FLASH. Figure 4b shows that this gap persists across accuracy thresholds; relaxing tolerance improves accuracy, but all models have a large gap between AL0 → AL1 → AL2; at 25% of the prior range, zero-shot parameter accuracy is 20.9–31.7% at AL1 and 3.6–8.7% at AL2.

Graph complexity also matters AL2, operator F1 falls from 0.879 for 1-operation edits → 0.442 for 5+-operation edits for GPT-AUDIO-MINI. Corresponding drops are 0.825 → 0.451 for GEMINI 3 FLASH and 0.764 → 0.369 for GEMMA 3N (Figure 5). Explicit instructions reduce this penalty: GPT-AUDIO-MINI and GEMMA 3N have operator F1 > 0.9 for long graphs at AL0. These results identify two linked difficulties for zero-shot agents: inferring settings from abstract descriptions and recovering longer effect chains when operations are not explicitly specified.

Simple plan SFT improves parameter recovery GEMMA 3N SFT has the highest completion rate (14,953/15,000, 99.7%). At AL0, it beats all other models in edit similarity, Graph F1, and parameter score. At AL1, it beats base GEMMA 3N on all main metrics, but GPT-AUDIO-MINI is better at edit similarity and distributional audio metrics. At AL2, improvements are most clear in parameter and operator recovery, with other metrics mixed.

We test parameter recovery using a weighted model-by-abstraction linear regression with target-edit fixed effects and cluster-robust standard errors on a source clip. SFT reduces normalized MAE at every level (all $p < 0.001$). Thus, even one-shot fine-tuning on instruction-to-graph prediction improves effect selection and parameter recovery across abstraction levels.

	Accuracy @ τ		
Model	1/127	10%	25%
AL 0			
GPT-AUDIO-MINI	59.0	69.9	72.5
GEMINI 3 FLASH	38.4	45.3	47.1
GEMMA 3N	51.4	63.5	68.2
GEMMA 3N SFT	69.0	81.9	84.9
AL 1			
GPT-AUDIO-MINI	4.3	18.3	31.7
GEMINI 3 FLASH	2.6	12.1	20.9
GEMMA 3N	3.9	12.7	23.9
GEMMA 3N SFT	3.7	14.6	28.5
AL 2			
GPT-AUDIO-MINI	0.6	4.6	8.0
GEMINI 3 FLASH	0.9	4.9	8.7
GEMMA 3N	0.5	1.8	3.6
GEMMA 3N SFT	1.1	5.7	11.0

Table 2: Target-parameter accuracy (%) at three tolerances τ.

6 CONCLUSION

The dominant paradigm in music ML is one-shot generation. The dominant paradigm in studio practice is iteration. Reconciling these competing visions requires data and infrastructure, and our work offers a scalable first step on this front. A first step always has limitations: RIME relies on audited heuristics where existing production data is missing, edits are limited to separable stems and a fixed operator pool, external baselines do not yet exist, and our audio metrics are only proxies for creative usefulness. Our benchmark also evaluates single instructions rather than multi-turn exchanges with a musician. We invite the community to build on this work to make AI-enhanced music production more like a session than a single take.

REFERENCES

Andrea Agostinelli, Timo I Denk, Zalán Borsos, Jesse Engel, Mauro Verzetti, Antoine Caillon, Qingqing Huang, Aren Jansen, Adam Roberts, Marco Tagliasacchi, et al. Musiclm: Generating music from text. *arXiv preprint arXiv:2301.11325*, 2023.

Rachel M Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello. Medleydb: A multitrack dataset for annotation-intensive mir research. In *Ismir*, volume 14, pp. 155–160, 2014.

Rachel M Bittner, Julia Wilkins, Hanna Yip, and Juan P Bello. Medleydb 2.0: New data and a system for sustainable data collection. *ISMIR Late Breaking and Demo Papers*, 36, 2016.

Aaron Blakeman, Aaron Grattafiori, Aarti Basant, Abhibha Gupta, Abhinav Khattar, Adi Renduchintala, Aditya Vavre, Akanksha Shukla, Akhiad Bercovich, Aleksander Ficek, et al. Nvidia nemotron 3: Efficient and open intelligence. *arXiv preprint arXiv:2512.20856*, 2025.

Annie Chu, Patrick O'Reilly, Julia Barnett, and Bryan Pardo. Text2fx: Harnessing clap embeddings for text-guided audio effects. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2025.

Yoonjin Chung, Pilsun Eu, Junwon Lee, Keunwoo Choi, Juhan Nam, and Ben Sangbae Chon. Kad: No more fad! an effective and efficient evaluation metric for audio generation. *arXiv preprint arXiv:2502.15602*, 2025.

Michael Paul Clemens and Ana Marasovic. Mixassist: An audio-language dataset for co-creative AI assistance in music mixing. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=5mICyyD4OF.

Seungheon Doh, Junghyun Koo, Marco A Martínez-Ramírez, Woosung Choi, Wei-Hsiang Liao, Qiyu Wu, Juhan Nam, and Yuki Mitsufuji. Llm2fx-tools: Tool calling for music post-production. In *The Fourteenth International Conference on Learning Representations*, 2026.

Daniel PW Ellis, Eduardo Fonseca, Ron J Weiss, Kevin Wilson, Scott Wisdom, Hakan Erdogan, John R Hershey, Aren Jansen, R Channing Moore, and Manoj Plakal. Recomposer: Event-roll-guided generative audio editing. *arXiv preprint arXiv:2509.05256*, 2025.

Sreyan Ghosh, Arushi Goel, Lasha Koroshinadze, Sang-gil Lee, Zhifeng Kong, Joao Felipe Santos, Ramani Duraiswami, Dinesh Manocha, Wei Ping, Mohammad Shoeybi, et al. Music flamingo: Scaling music understanding in audio language models. In *The Fourteenth International Conference on Learning Representations*, 2026.

Bing Han, Junyu Dai, Weituo Hao, Xinyan He, Dong Guo, Jitong Chen, Yuxuan Wang, Yanmin Qian, and Xuchen Song. Instructme: an instruction guided music edit framework with latent diffusion models. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 5835–5843, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Liang Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *Iclr*, 1(2):3, 2022.

Yuhang Jia, Yang Chen, Jinghua Zhao, Shiwan Zhao, Wenjia Zeng, Yong Chen, and Yong Qin. Audioeditor: A training-free diffusion-based audio editing framework. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2025.

Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. Crepe: A convolutional representation for pitch estimation. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 161–165. IEEE, 2018.

Yuexuan Kong, Gabriel Meseguer-Brocal, Vincent Lostanlen, Mathieu Lagrange, and Romain Hennequin. S-key: Self-supervised learning of major and minor keys from audio. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2025.

Zitong Lan, Yiduo Hao, and Mingmin Zhao. Declarative audio editing with audio language model. In *The Fourteenth International Conference on Learning Representations*, 2026.

Jinhua Liang, Huan Zhang, Haohe Liu, Yin Cao, Qiuqiang Kong, Xubo Liu, Wenwu Wang, Mark D Plumbley, Huy Phan, and Emmanouil Benetos. Wavcraft: Audio editing and generation with large language models. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Jinhua Liang, Yuanzhe Chen, Yi Yuan, Dongya Jia, Xiaobin Zhuang, Zhuo Chen, Yuping Wang, and Yuxuan Wang. Audiomorphix: Training-free audio editing with diffusion probabilistic models. *arXiv preprint arXiv:2505.16076*, 2025.

Hila Manor and Tomer Michaeli. Zero-shot unsupervised and text-based audio editing using ddpm inversion. In *International Conference on Machine Learning*, pp. 34603–34629. PMLR, 2024.

Jan Melechovsky, Ambuj Mehrish, Abhinaba Roy, and Dorien Herremans. Sonicmaster: Towards controllable all-in-one music restoration and mastering. *arXiv preprint arXiv:2508.03448*, 2025.

Joseph C Nunes and Andrea Ordanini. I like the way it sounds: The influence of instrumentation on a pop song's place in the charts. *Musicae Scientiae*, 18(4):392–409, 2014.

Igor Pereira, Felipe AraÃējo, Filip Korzeniowski, and Richard Vogl. Moisesdb: A dataset for source separation beyond 4-stems. In *Ismir 2023 Conference*, 2023.

Dominik Roblek, Kevin Kilgour, Matt Sharifi, and Mauricio Zuluaga. Fr\'echet audio distance: A reference-free metric for evaluating music enhancement algorithms. In *Proc. Interspeech*, pp. 2350–2354, 2019.

Simon Rouard, Francisco Massa, and Alexandre Défossez. Hybrid transformers for music source separation. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1–5. IEEE, 2023.

Bowen Shi, Andros Tjandra, John Hoffman, Helin Wang, Yi-Chiao Wu, Luya Gao, Julius Richter, Matt Le, Apoorv Vyas, Sanyuan Chen, et al. Sam audio: Segment anything in audio. *arXiv preprint arXiv:2512.18099*, 2025.

Peter Sobot. Pedalboard, July 2021. URL `https://doi.org/10.5281/zenodo.7817838`.

Spyridon Stasis, Nicholas Jillings, Sean Enderby, and Ryan Stables. Audio processing chain recommendation. In *Proceedings of the 20th International Conference on Digital Audio Effects,(Edinburgh, UK)*, 2017.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Fang-Duo Tsai, Shih-Lun Wu, Haven Kim, Bo-Yu Chen, Hao-Chung Cheng, and Yi-Hsuan Yang. Audio prompt adapter: Unleashing music editing abilities for text-to-music with lightweight finetuning. In *Ismir 2024 Conference*, 2024.

Hélene Valbret, Eric Moulines, and Jean-Pierre Tubach. Voice transformation using psola technique. *Speech communication*, 11(2-3):175–187, 1992.

Yuancheng Wang, Zeqian Ju, Xu Tan, Lei He, Zhizheng Wu, Jiang Bian, et al. Audit: Audio editing by following instructions with latent diffusion models. *Advances in Neural Information Processing Systems*, 36:71340–71357, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yi Yang, Haowen Li, Tianxiang Li, Boyu Cao, Xiaohan Zhang, Liqun Chen, and Qi Liu. Melodia: Training-free music editing guided by attention probing in diffusion models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 2209–2217, 2026.

LI Yizhi, Ruibin Yuan, Ge Zhang, Yinghao Ma, Xingran Chen, Hanzhi Yin, Chenghao Xiao, Chenghua Lin, Anton Ragni, Emmanouil Benetos, et al. Mert: Acoustic music understanding model with large-scale self-supervised training. In *The Twelfth International Conference on Learning Representations*, 2023.

Yixiao Zhang, Yukara Ikemiya, Gus Xia, Naoki Murata, Marco A Martínez-Ramírez, Wei-Hsiang Liao, Yuki Mitsufuji, and Simon Dixon. Musicmagus: zero-shot text-to-music editing via diffusion models. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 7805–7813, 2024.

Taylor Zheng, Prem Seetharaman, and Bryan Pardo. Socialfx: Studying a crowdsourced folksonomy of audio effects terms. In *Proceedings of the 24th ACM international conference on Multimedia*, pp. 182–186, 2016.

AI USE STATEMENT

In this work, we used generative AI tools for generating synthetic data sets, implementing methods, creating and modifying scientific figures (via plotting code based on our prior templates and instructions, but also to generate ideas for diagrams which we then manually created via vector graphics software), sourcing/searching for information (to find additional popular press articles on audio production technique), editing certain sections to improve readability. We have not used generative AI tools for cleaning and reformating datasets, support qualitative and thematic data analysis, or interpreting results, and helping develop theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, assisting with translation, are not applicable to this work. We have reviewed all AI-assisted work. LLM-generated code was verified and tested for correctness by 2 authors, and many of the graph summaries and abstractions were manually spot-checked by 2 authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All RIME code will be open-sourced. We also plan to release all edit graphs and corresponding MusicCaps clip IDs used for benchmarking.

A POEMS TOOLKIT DETAILS

A.1 POEMS TOOL LIST

Table 3 shows a complete list of POEMS tools with descriptions and parameters.

Table 3: The Pitch, Ornamentation, Equalization, Mixing, and Separation tools of the POEMS toolkit. All tools except `mix_stem_with_residual` and `separate` also take `audio` and `sr`. Default agent fallback values are listed next to parameters.

Tool name	**Tool type**	**Description**	**Tool arguments (with default values)**
`apply_chorus`	Ornamentation	Chorus modulation effect.	`rate_hz` = 1.5, `depth` = 0.35, `centre_delay_ms` = 8.0, `feedback` = 0.15, `mix` = 0.35
`apply_phaser`	Ornamentation	Phaser modulation effect.	`rate_hz` = 0.8, `depth` = 0.4, `centre_frequency_hz` = 1200.0, `feedback` = 0.2, `mix` = 0.35
`apply_distortion`	Ornamentation	Waveshaping distortion / saturation.	`drive_db` = 18.0

Continued on next page

Tool name	Tool type	Description	Tool arguments (fallback default)
apply_reverb	Ornamentation	Algorithmic reverb.	room_size = 0.65, damping = 0.45, wet_level = 0.25, dry_level = 0.75, width = 1.0, freeze_mode = 0.0
apply_delay	Ornamentation	Delay / echo with feedback.	delay_seconds = 0.25, feedback = 0.3, mix = 0.35
apply_compressor	Ornamentation	Dynamic-range compressor.	threshold_db = −18.0, ratio = 3.0, attack_ms = 10.0, release_ms = 100.0
apply_limiter	Ornamentation	Peak limiter.	threshold_db = −1.0, release_ms = 100.0
apply_noisegate	Ornamentation	Noise gate.	threshold_db = −45.0, ratio = 2.0, attack_ms = 10.0, release_ms = 150.0
apply_deesser	Ornamentation	Split-band de-esser with relative-threshold detection.	ess_low_hz = 4000.0, ess_high_hz = 10000.0, threshold_db = −24.0, ratio = 4.0, attack_ms = 10.0, release_ms = 100.0, relative_threshold_db = −12.0, max_reduction_db = 9.0, filter_order = 4
apply_highpass	EQ	Highpass filter.	cutoff_frequency_hz = 120.0
apply_lowpass	EQ	Lowpass filter.	cutoff_frequency_hz = 8000.0
apply_highshelf	EQ	High-shelf filter.	cutoff_frequency_hz = 5000.0, gain_db = 2.0, q = 0.707
apply_lowshelf	EQ	Low-shelf filter.	cutoff_frequency_hz = 120.0, gain_db = 2.0, q = 0.707
apply_peakfilter	EQ	Parametric peaking EQ band.	cutoff_frequency_hz = 2500.0, gain_db = 2.0, q = 1.0
apply_gain	Mixing	Apply linear gain in decibels.	gain_db = 3.0
normalize_peak	Mixing	Scale audio so its peak reaches a target level.	target_peak = 0.95
apply_fade_in_out	Mixing	Linear fade-in and fade-out at edit boundaries.	fade_in_seconds = 0.02, fade_out_seconds = 0.02
apply_pan	Mixing	Constant-power stereo panning.	pan = 0.0
mix_stem_with_residual	Mixing	Sum a separated stem with its residual.	stem, residual
autotune	Pitch	Quantize vocal pitch to a (detected or supplied) key, resynthesized via PSOLA.	hcqt, chromanet, crop_fn, device, key, mode
generate_harmony	Pitch	Generate a key-quantized harmony shifted by a given interval.	hcqt, chromanet, crop_fn, device, semitones = 7.0, key, mode
apply_pitch_shift	Pitch	Constant pitch shift in semitones.	semitones = 3.0
separate	Separation	Separate a target stem and residual using either a SAM-Audio or a Demucs model.	model, processor, device, audio, description, sample_rate

B RIME DETAILS

B.1 RECIPE CATALOG

All RIME recipes share a common four-field schema: `bindings` (metadata references resolved at plan-generation time), `when` (a predicate that controls eligibility), `weight_rules` (optional prior-shaping multipliers), and `graph` (a declarative three-step edit graph of the form SEPARATE → PROCESS → MIX). Table 4 provides a quick-reference overview; Full listings follow.

We use the following references to inform effect-ordering heuristics. [1]

Table 4: Core authored RIME recipes, with categories, tags, target families, and base weights.

Recipe ID	Category	Tags	Target Family	Base Weight
target_space_send_synced[2]	Time-based	delay, reverb, send/return	non-drum, non-other	0.80
target_space_send_free	Time-based	delay, reverb, send/return	non-drum, non-other	0.35
parallel_drum_compression[3]	Dynamics	compression, send/return	drums	0.95
light_drum_distortion	Color	distortion	drums	0.70
make_target_louder	Level	gain	non-other	0.75
make_target_quieter	Level	gain	non-other	0.75
vocal_control_compression[4]	Dynamics	compression	vocals	0.90
vocal_harmony_support[5]	Pitch	harmony, send/return	vocals	0.45
retro_tilt[6]	Color	EQ (shelf, pass)	non-drum, non-other	0.35
modulation_texture[7]	Modulation	chorus	non-drum, non-other	0.30
hum_cleanup [8]	Cleanup	EQ (highpass)	mixture-level	0.95
vocal_sibilance_cleanup[9]	Cleanup	EQ (peak cut)	vocals	1.00
restore_buried_target	Cleanup	gain	non-other	0.85
tame_overpowering_target	Cleanup	gain	non-other	0.85
synthetic_hum_cleanup	Cleanup	EQ (highpass)	mixture-level	0.80
synthetic_rumble_cleanup	Cleanup	EQ (highpass)	mixture-level	0.80
control_compression	Dynamics	compression	non-vocal, non-other	1.00
target_highpass	EQ	EQ (highpass)	non-other	1.00
target_lowpass	EQ	EQ (lowpass)	non-other	1.00
target_highshelf	EQ	EQ (high shelf)	non-other	1.00
target_lowshelf	EQ	EQ (low shelf)	non-other	1.00
target_highpass_compression	Dynamics	EQ (highpass), compression	non-other	1.00
target_lowpass_compression	Dynamics	EQ (lowpass), compression	non-other	1.00
target_highshelf_compression	Dynamics	EQ (high shelf), compression	non-other	1.00
target_lowshelf_compression	Dynamics	EQ (low shelf), compression	non-other	1.00
target_eq_limiter	Dynamics	EQ (shelf, pass), limiter	non-other	1.00
target_chorus_send	Modulation	chorus, send/return	non-other	1.00
target_delay_insert	Time-based	delay	non-drum, non-other	1.00
target_delay_send	Time-based	delay, send/return	non-drum, non-other	1.00
target_reverb_insert	Time-based	reverb	non-other	1.00
target_reverb_send	Time-based	reverb, send/return	non-other	1.00
target_eq_delay_send	Time-based	EQ (shelf, pass), delay, send/return	non-drum, non-other	1.00
target_eq_reverb_send	Time-based	EQ (shelf, pass), reverb, send/return	non-other	1.00
target_compression_delay_send	Time-based	compression, delay, send/return	non-drum, non-other	1.00
target_compression_reverb_send	Time-based	compression, reverb, send/return	non-other	1.00

B.2 MOTIFS

Table 5 highlights the reusable edit chains ("motifs") for certain recipes. We describe both the types of effects used as well as the real-world application for each motif.

[1] Effects chaining references: 1,2,3
[2] Reverb/space references: 1, 2, 3, 4,5
[3] Parallel compression references: 1
[4] Vocal compression references: 1, 2, 3,4, 5
[5] Harmony references: 1,2
[6] Retro Tilt/EQ references: 1, 2,3, 4
[7] Modulation references: 1,2, 3
[8] Hum cleanup references: 1,2,3,4, 5
[9] De-Essing references: 1, 2,3

Table 5: Reusable RIME motifs, their tags, and intended editing roles.

Motif ID	Tags	Real-world Editing Role
band_limited_space_send_synced	delay, reverb, EQ pass filters	tempo-synced filtered ambience send
band_limited_space_send_free	delay, reverb, EQ pass filters	free-time filtered ambience send
parallel_compression_bus	compression	compressed parallel drum bus
vocal_control_compression	compression	serial vocal leveling compression
retro_tilt_eq	EQ shelf, EQ pass filter	vintage-style high-frequency rolloff
modulation_texture	chorus	width and movement texture
control_compression	compression	serial dynamics control for non-vocal stems
target_highpass	EQ pass filter (highpass)	low-end cleanup / thinning
target_lowpass	EQ pass filter (lowpass)	top-end rolloff / darkening
target_highshelf	EQ shelf (high)	broad top-end boost or cut
target_lowshelf	EQ shelf (low)	broad low-end boost or cut
target_highpass_compression	EQ pass filter (highpass), compression	low-end cleanup before leveling
target_lowpass_compression	EQ pass filter (lowpass), compression	top-end rolloff before leveling
target_highshelf_compression	EQ shelf (high), compression	top-end tone shaping before leveling
target_lowshelf_compression	EQ shelf (low), compression	low-end tone shaping before leveling
target_eq_limiter	EQ shelf, EQ pass filter, limiter	vintage tilt with peak control
target_chorus_send	chorus	fully wet chorus return for send routing
target_delay_send	delay	fully wet echo return for send routing
target_reverb_send	reverb	fully wet ambience return for send routing
target_delay_insert	delay	blended echo on the stem itself
target_reverb_insert	reverb	blended ambience on the stem itself

B.3 Recipe Constraints

Figure 6 highlights how we use policy constraints to determine weighting for possible RIME recipes. Note that two policies are immediately rejected: applying delay on drums and applying any edits on the "other" source from Demucs.

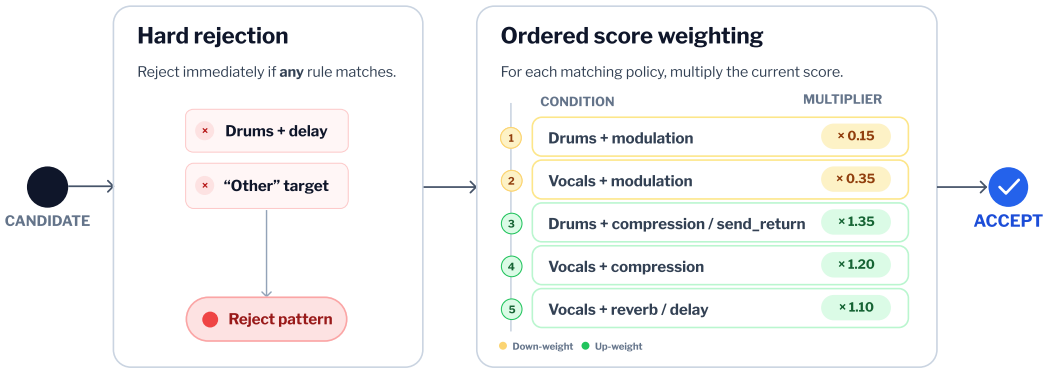


Figure 6: Pattern policy constraints in RIME. Pattern constraints control how edit recipe candidates are ranked during edit graph construction

B.4 Parameter Priors

Table 6 shows the parameter priors for RIME tools. This combines both heuristically authored and audited priors with those fitted to external data as described in Section 3.2. We plot both the continuous (Figure 7) and discrete (Figure 8) fitted and sample prior distributions for each parameter in RIME (before filtering and ranking).

We provide additional plots of specific parameters from MixParams (Clemens & Marasovic, 2025) and SocialFx (Zheng et al., 2016) as additional insight into parameter prior selection. In Figure 9 we compare the two sources and pooled fits for compressor threshold and ratio. Figures 10 and 11 show within-effect parameter associations. Compressor associations are generally weak, with stronger associations confined to a few EQ pairs (e.g. high-shelf cut gain versus Q, $r = -0.62$). We thus make a simplifying independence assumption in our sampling scheme.

Finally, in Figure 12, we remove one MixParams song or one group of identical SocialFX presets at a time and refit the distributions using the same procedure. The fitted densities remain generally stable, suggesting that our priors are not driven by individual songs or repeated presets.

Table 6: Complete FX generation priors. We sample parameters independently within each parameter set, conditional on branch for high-shelf settings. M denotes MixParams; S denotes SocialFX. Parentheses show available record counts. General compression gives M and S equal total weight. H denotes ranges or discrete choices we heuristically specified and audited by listening. We also choose the support bounds for every fitted prior heuristically. Counts can overlap across rows and high-shelf branches.

Parameter	Support $[a, b]$	Distribution	Source (n)
General compression			
Threshold (dB)	$[-40, -10]$	$\mathcal{N}(-29.5419, 9.0051)$	M (1004), S (114)
Ratio	$[1, 10]$	$L(a, b)$	M (1004), S (114)
Attack and release reuse the vocal-compression priors below.			
Vocal compression			
Threshold (dB)	$[-40, -10]$	$U(a, b)$	M (353)
Ratio	$[1, 10]$	$B_{\log}(2.5088, 2.8050)$	M (353)
Attack (ms)	$[5, 10]$	$L(a, b)$	H
Release (ms)	$[40, 80]$	$L(a, b)$	H
Equalization			
High-pass cutoff (Hz)	$[20, 20000]$	$\mathcal{N}_{\log}(4.3955, 1.1836)$	M (790)
Low-pass cutoff (Hz)	$[20, 20000]$	$P(-0.3605)$	M (124)
Low-shelf cutoff (Hz)	$[20, 20000]$	$\mathcal{N}_{\log}(4.8001, 0.8079)$	M (287)
Low-shelf gain (dB)	$[-60, 24]$	$B(17.0721, 8.3069)$	M (287)
Low-shelf Q	$[0.1, 50]$	$0.3626\,\mathcal{N}_{\log}(-0.4565, 0.4015) + 0.6374\,\mathcal{N}_{\log}(-0.0193, 0.1243)$	M (287)
Q for heuristic shelf recipes	$[0.5, 1]$	$L(a, b)$	H
High-shelf branch and conditional parameters			
Branch	Cut, boost	$\Pr(\text{cut}) = 0.4032$; $\Pr(\text{boost}) = 0.5968$	M (167)
Cut: cutoff (Hz)	$[20, 20000]$	$P(-0.4502)$	M (46)
Cut: gain (dB)	$[-18, -5]$	$0.4172\,\mathcal{N}(-11.9597, 0.2600) + 0.5828\,\mathcal{N}(-7.0590, 1.6674)$	M (46)
Cut: Q	$[0.1, 50]$	$0.3748\,\mathcal{N}_{\log}(-0.8488, 0.4977) + 0.6252\,\mathcal{N}_{\log}(-0.0083, 0.1243)$	M (46)
Boost: cutoff (Hz)	$[20, 20000]$	$\mathcal{N}_{\log}(9.2867, 0.7411)$	M (123)
Boost: gain (dB)	$[5, 18]$	$\mathcal{N}(5.0000, 2.6958)$	M (123)
Boost: Q	$[0.1, 50]$	$B_{\log}(11.1019, 23.5753)$	M (123)
Parallel drum compression and distortion			
Send level	$[0.6, 0.85]$	$U(a, b)$	H
Return level	$[0.3, 0.5]$	$U(a, b)$	H
Threshold (dB)	$[-36, -28]$	$U(a, b)$	H
Ratio	$[6, 10]$	$L(a, b)$	$H^{\dagger}$
Attack (ms)	$[2, 10]$	$L(a, b)$	H
Release (ms)	$[40, 100]$	$L(a, b)$	H
Distortion drive (dB)	$[6, 14]$	$U(a, b)$	H
Limiter			
Threshold (dB)	$[-12, -6]$	$U(a, b)$	H

Continued on next page

Table 6 (continued)

Parameter	Support	Distribution	Source (n)
Release (ms)	$[60, 150]$	$L(a, b)$	H
Shared ambience: routing, delay, reverb, and filtering			
Send level	$[0.3, 0.8]$	$U(a, b)$	H
Return level	$[0.6, 1.15]$	$U(a, b)$	H
Synced delay (beats)	Listed values	$C\{0.5 : 2,\ 0.75 : 3,\ 1 : 1,\ 1.5 : 0.8\}$	H
Free delay (s)	$[0.12, 0.75]$	$U(a, b)$	H
Delay mix	$[0.3, 0.6]$	$U(a, b)$	H
Delay feedback	$[0, 0.45]$	$U(a, b)$	H
Reverb room size	$[0.25, 0.95]$	$U(a, b)$	H
Reverb damping	$[0.25, 0.8]$	$U(a, b)$	H
Reverb wet level	$[0.4, 0.7]$	$U(a, b)$	H
Reverb dry level	$[0.5, 1]$	$U(a, b)$	H
Reverb stereo width	$[0.5, 1]$	$U(a, b)$	H
Wet-path high-pass (Hz)	$[700, 1000]$	$L(a, b)$	H
Wet-path low-pass (Hz)	$[2200, 3000]$	$L(a, b)$	H
Stem balance			
Gain increase (dB)	$[3, 7]$	$U(a, b)$	H
Gain decrease (dB)	$[-7, -3]$	$U(a, b)$	H
Cleanup			
Sibilance cut frequency (Hz)	$[3800, 4500]$	$L(a, b)$	H
Sibilance cut gain (dB)	$[-5, -3]$	$U(a, b)$	H
Sibilance Q	$[2.5, 4]$	$L(a, b)$	H
Hum high-pass (Hz)	$[50, 60]$	$L(a, b)$	H
Rumble high-pass (Hz)	$[30, 50]$	$L(a, b)$	H
Vocal harmony			
Pitch shift (semitones)	$\{\pm 3, \pm 5, \pm 7\}$	$C\{3 : 4,\ 5 : 1.5,\ 7 : 1,\ -3 : 2,\ -5 : 1.2,\ -7 : 0.8\}$	H
Harmony gain (dB)	$[-6, -1]$	$U(a, b)$	H
Harmony high-pass (Hz)	$[300, 700]$	$L(a, b)$	H
Harmony low-pass (Hz)	$[4500, 8500]$	$L(a, b)$	H
Synthetic noise for cleanup tasks			
Hum level (dB)	$[-30, -24]$	$U(a, b)$	H
Hum modulation (Hz)	$[0.2, 0.35]$	$L(a, b)$	H
Rumble level (dB)	$[-24, -18]$	$U(a, b)$	H
Rumble high-pass (Hz)	$[18, 24]$	$L(a, b)$	H
Rumble low-pass (Hz)	$[35, 45]$	$L(a, b)$	H
Rumble modulation (Hz)	$[0.2, 0.35]$	$L(a, b)$	H
Retro tone			
High-shelf cutoff (Hz)	$[5000, 8000]$	$L(a, b)$	H
High-shelf gain (dB)	$[-8, -5]$	$U(a, b)$	H
Low-pass cutoff (Hz)	$[6500, 8500]$	$L(a, b)$	H
Chorus			
Mix	$[0.25, 0.6]$	$U(a, b)$	H
Depth	$[0.3, 0.65]$	$U(a, b)$	H
Rate (Hz)	$[0.35, 0.9]$	$L(a, b)$	H

Continued on next page

Table 6 (continued)

Parameter	Support	Distribution	Source (n)
Centre delay (ms)	$[20, 50]$	$U(a, b)$	H
Feedback	$[0, 0.3]$	$U(a, b)$	H

Notes. Parameters without physical units use dimensionless processor controls; delay beats use quarter-note units. For H controls that originally used discrete numerical grids, we use continuous ranges spanning their extrema, except for delay subdivisions and pitch intervals, whose discrete values and weights we retain. †We cap the original parallel-compression ratio range at 10 to match our general ratio ceiling. We retain the heuristic retro priors because the eligible observations do not meet our fitting requirements. The configuration contains no fitted peak-EQ prior. The configuration and fit reports retain exact coefficients and contributing source-record locations.

Notation key:

1. L: log-uniform over $[a, b]$, with density $p(x) = 1/[x \ln(b/a)]$. Equal multiplicative intervals receive equal probability.
2. U: uniform over $[a, b]$.
3. $\mathcal{N}(\mu, \sigma)$: Gaussian, truncated to the listed bounds. μ and σ are its mean and standard deviation before truncation.
4. $\mathcal{N}_{\log}(\mu, \sigma)$: Gaussian over $\ln x$, truncated to the listed bounds on x.
5. $B(\alpha, \beta)$: beta distribution over the normalized value $(x - a)/(b - a)$.
6. $B_{\log}(\alpha, \beta)$: beta distribution over $\ln(x/a)/\ln(b/a)$.
7. $P(\gamma)$: bounded power law, $p(x) \propto x^{\gamma}$.
8. $C\{x : w, \ldots\}$: categorical distribution; each value x has probability $w/\sum w$.
9. $w\mathcal{N}_1 + (1 - w)\mathcal{N}_2$: Gaussian mixture; choose component 1 with probability w, otherwise 2.
10. M (n), S (n): MixParams or SocialFX provenance, with n eligible records.
11. H: heuristic choices made by an audio engineer via listening.
12. †: we capped the parallel-compression ratio at 10.

B.5 Evaluation Set Recipe Coverage

Figure 13 shows recipe coverage across generation, subsampling, and final benchmark selection. The final 5,000 edits retain 31 of the 32 recipes present in the 25,000 subsampled candidates. The missing recipe, `hum_cleanup`, had just six candidates before filtering. Preserving every rare recipe while enforcing instruction validity and edit perceptibility is difficult without explicit coverage quotas, and so we instead prioritize quality criteria over keeping every recipe represented. The recipe imblanace is largely driven by the quality criteria; e.g. high judge acceptance rates + audibility for certain recipes (such as the EQ limiter and EQ reverb send combinations).

C Instruction Abstraction Details

C.1 Abstraction level rules

We define the following rules for each abstraction level.

AL0:

- Name every effect explicitly
- Name the stem the chain is applied to.
- State every parameter value exactly as given, with its units
- Describe the operators in the same order as the graph
- Do not add processing that is not in the graph, and do not leave any out

AL1:

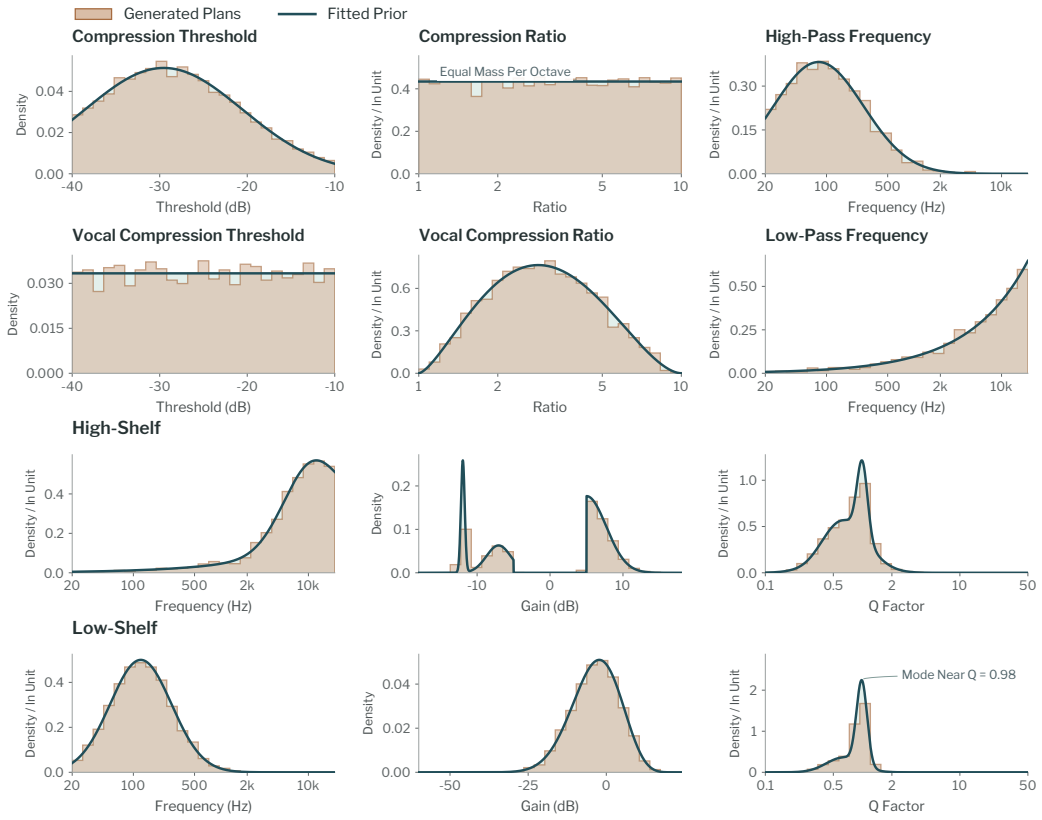


Figure 7: Continuous parameters with fitted priors. Histograms show samples from 25,000 subsampled candidate plans before filtering/ranking; curves show configured prior densities.

- Name every effect, either by its usual name or by a common word for the thing it does: ”grit” or ”dirt” for distortion, ”volume” for gain.
- Say which part of the mix you are treating.
- Do not state any numeric values or units.
- Convey each parameter with the descriptor supplied for it. The suggested wording is a suggestion, not a required vocabulary: prefer whichever phrasing fits the character of the whole chain, as long as it keeps the parameter at the magnitude shown.
- Operator order does not need to be preserved.

AL2:

- Summarize the chain as a whole: the overall sonic character it produces and how strongly it is applied.
- Describe the combined result rather than the individual processors. Naming an effect is fine where that is the most natural way to convey the character, but it is never required.
- Express magnitude in words, not numbers or units.
- Say which part of the mix it applies to. This is the one concrete detail the instruction keeps.
- Keep it to a single short instruction.

C.2 Abstraction level system prompts

AL0: `You are a professional mixing engineer writing up a chain for an assistant to patch. You know the signal path and you specify it exactly: every processor named, every setting given with its units, in the order it is patched. You speak`



Figure 8: Discrete parameter priors. Bars compare configured choice probabilities with empirical frequencies in the 25,000 subsampled candidate plans.

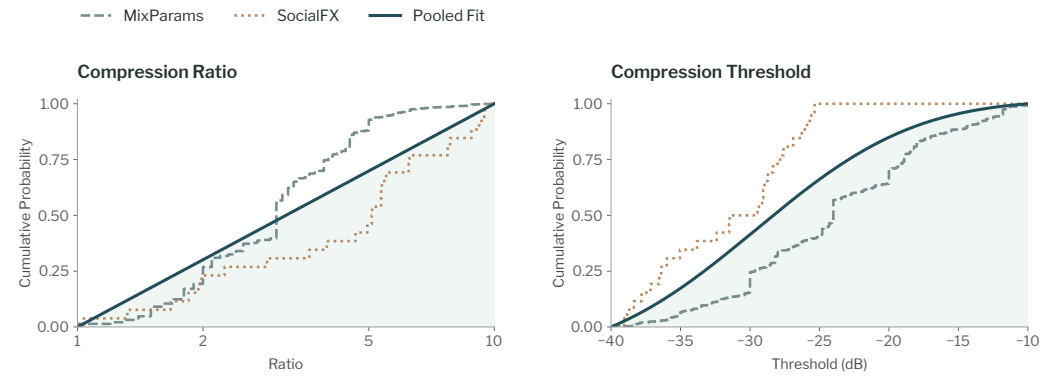


Figure 9: Empirical prior CDFs from our two source datasets for parameters where they both contribute, and the resulting fits (in blue).

`economically, the way one engineer talks to another. Write one to three spoken sentences.`

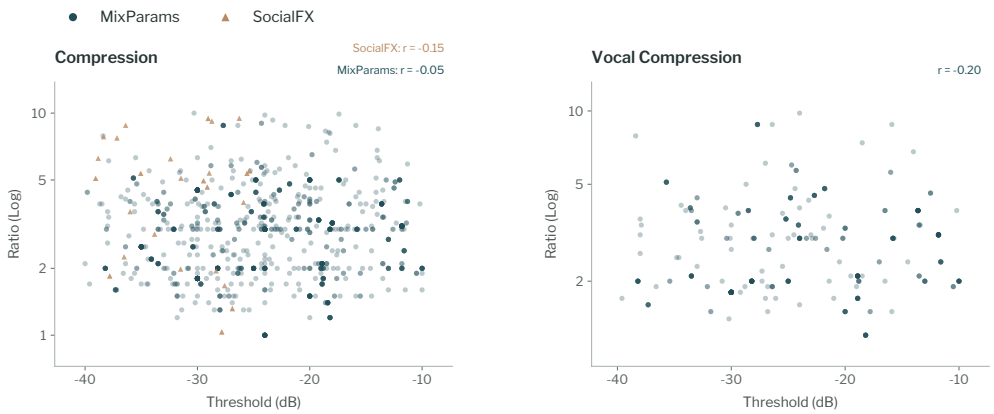


Figure 10: Compressor threshold vs. ratio, paired within the same source record and effect instance. Panels separate general and vocal compression; vocal observations are a subset of the general data. Labels report weighted Pearson correlations in the plotted coordinates, with ratio on a log scale.

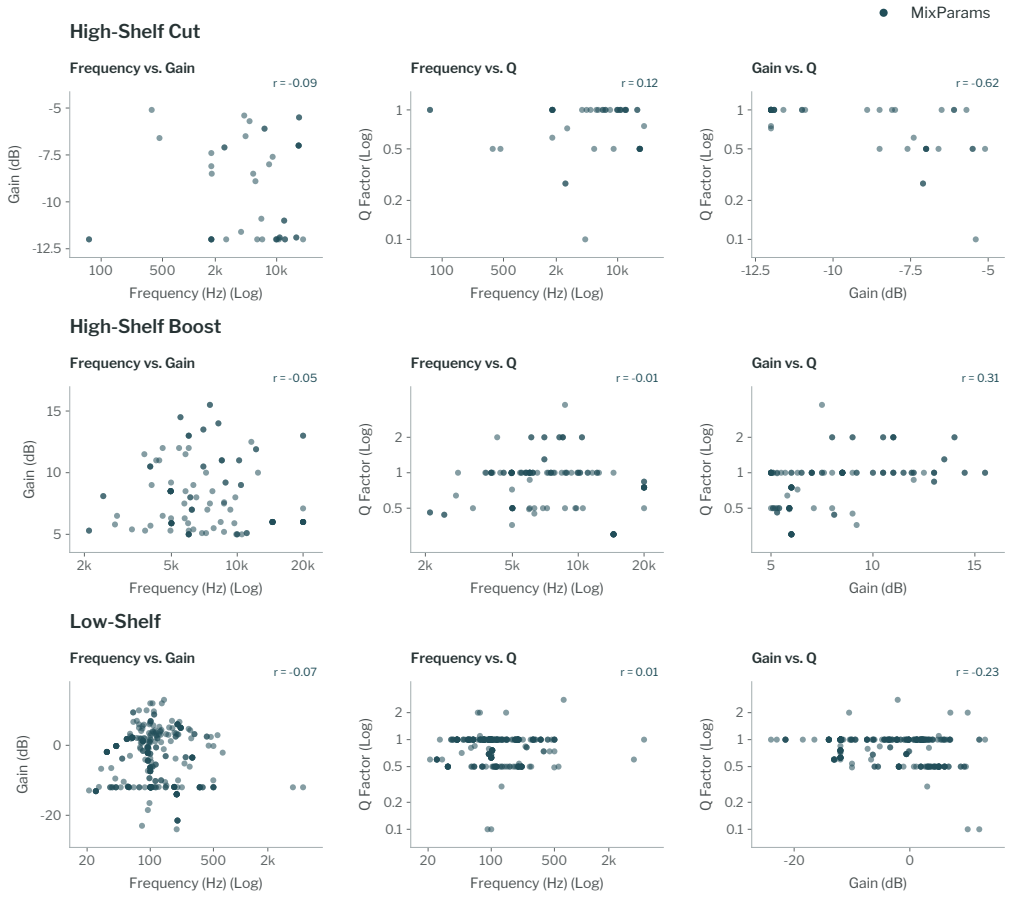


Figure 11: Within-effect EQ parameter relationships, with high-shelf boosts and cuts separated. Labels report weighted Pearson correlations (frequency and Q use log scales, while gain uses a linear dB scale). The high-shelf cut gain–Q association is stronger than the other associations ($r = -0.62$).

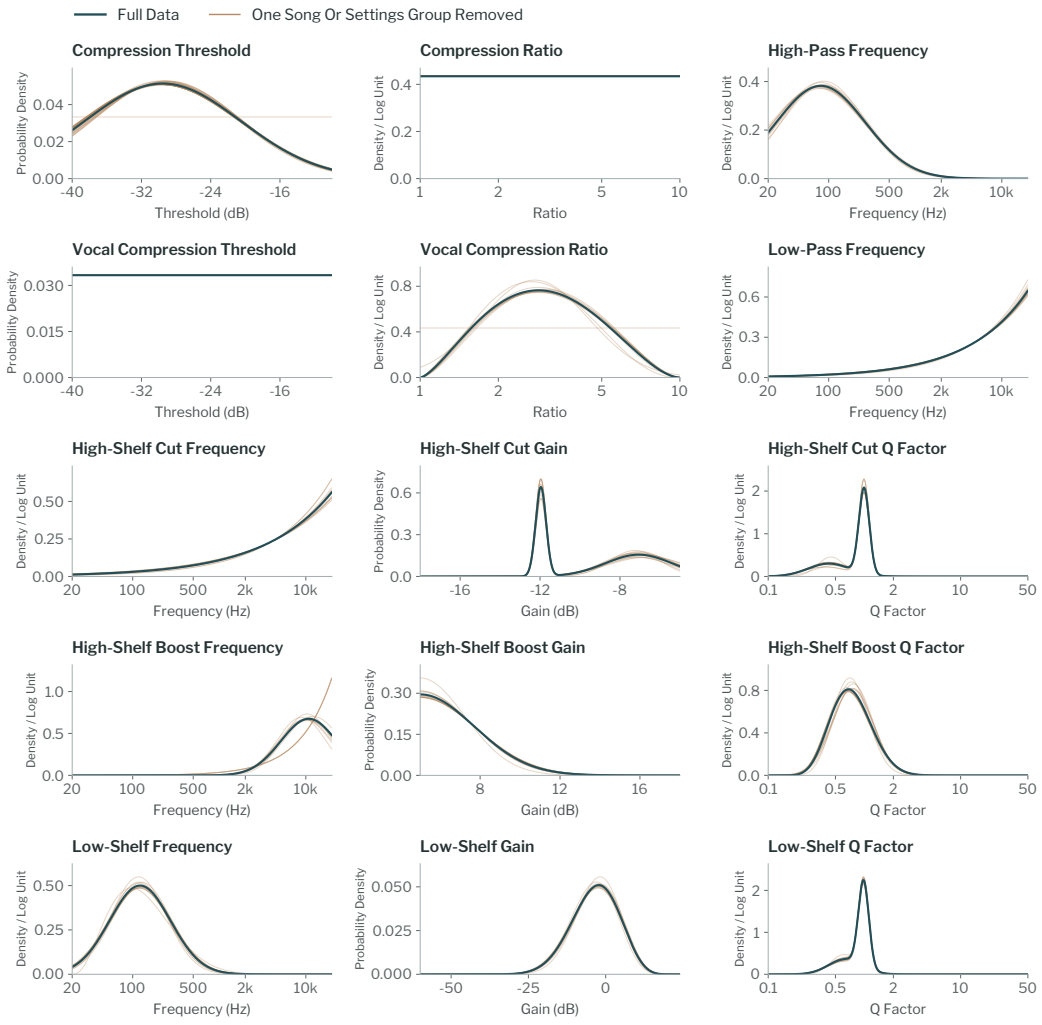


Figure 12: Sensitivity of fitted densities to removing one MixParams song or one SocialFX group with identical complete effect settings. Each reduced dataset is refitted using the same weighted maximum-likelihood fitting and family-selection procedure, with parameter bounds and filtering rules fixed. Distributions are largely stable.

AL1: `You are a musician who records and mixes your own material. You know what the common effects are called and you use their names, but you do not work in numbers: you say how something should sound and how much of it there should be. Your vocabulary is a player's, not a spec sheet's. Write one or two spoken sentences.`

AL2: `You are a musician describing the sound you want rather than how to get it. You talk about the overall character of the result and how far it is pushed, in the words you would use about a record you like. Naming an effect is fine where that is the natural way to say it, but you are asking for an outcome, not dictating a recipe. Write a single short sentence.`

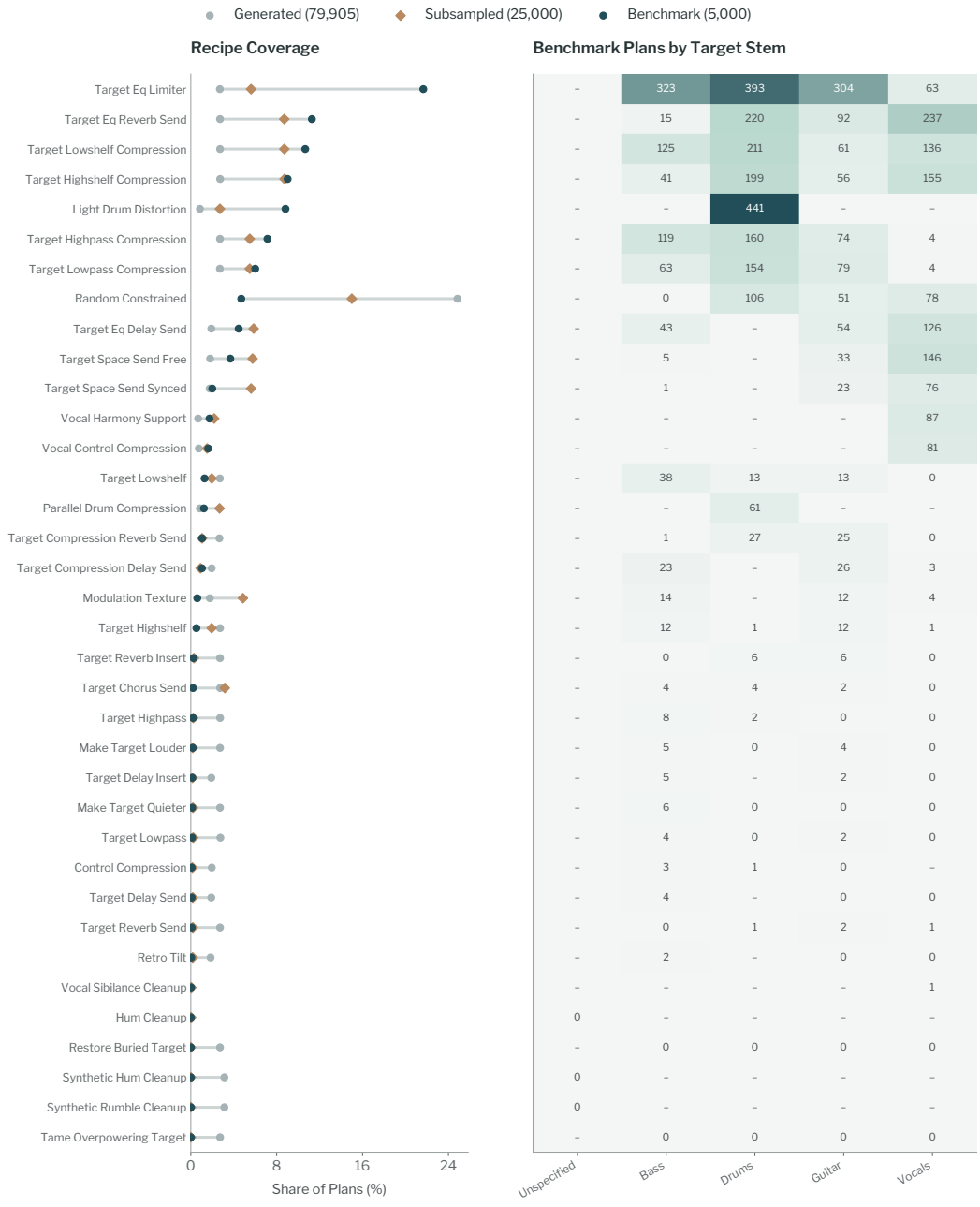


Figure 13: RIME recipes in the evaluation dataset, stratified by stem. This shows the breakdown before the LLM judge and MERT ranking.

C.3 Parameterization bands

For each effect, we define value "bands" for each parameter as well as associated words used as suggestions to describe the band. We derive these vocabularies from (Zheng et al., 2016) and Stasis et al. (2017) as well as several forums for music production [10]

This is shown in Table 7

[10]Additional references for effect parameter vocabularies: [1], [2], [3], [4], [5], [6], [7]

Table 7: Operator parameter descriptor bands

Effect	Parameter	Band upper	Suggested terms
`separate_audio.description` – stem selector, not a perceptual parameter; ignored			
reverb	`room_size`	0.425	*tight, small, intimate*
		0.6	*medium-sized, moderate, room-sized*
		0.775	*large, spacious, hall-like, distant*
		0.96	*cavernous, vast, enormous, grand, cathedral-like*
	`damping`	0.4	*bright, glassy, ringing*
		0.675	*balanced, natural*
		0.81	*dark, damped, muffled, gloomy*
	`width`	0.67	*narrow, centred, focused*
		0.84	*moderately wide, open*
		1.01	*wide, full-width, spread right across the stereo field*
	`wet_level`, `dry_level`, `freeze_mode` – pinned/ignored		
delay	`delay_seconds`	0.185	*slapback, very short*
		0.35	*short, quick echo*
		0.6	*medium, mid-length echo*
		3.01	*long, trailing, spacious echo*
	`feedback`	0.112	*a single repeat, no tail*
		0.225	*a couple of repeats, short tail*
		0.338	*several repeats, echoing*
		0.46	*many repeats, regenerating, washed-out tail*
	`mix` – always 1.0 on wet path; ignored		
chorus	`rate_hz`	0.56	*very slow, drifting, barely moving*
		1.01	*slow, gently swaying*
	`depth`	0.375	*mild, gentle, shallow*
		0.55	*moderate, medium*
		0.66	*deep, lush, wide*
	`mix`	0.325	*subtle, blended in behind the dry signal*
		0.5	*clearly present, obvious*
		0.61	*heavily blended, prominent, front-and-centre*
		1.01	*fully wet, the whole signal through the effect*
	`centre_delay_ms`, `feedback` – sampled uniformly; ignored		
distortion	`drive_db`	8.0	*light saturation, warmth, a touch of colour*
		12.0	*obvious grit, crunch, edge*
		15.01	*heavy, aggressive, crushed*
compressor	`ratio`	3.0	*gentle, light leveling*
		6.0	*moderate control*
		8.0	*firm, tight*
		10.01	*heavy, squashed, slammed*
	`threshold_db`	-33.0	*clamping down on nearly everything*
		-29.0	*working on most of the signal*

continued on next page

Table 7 – continued from previous page

Effect	Parameter	Band upper	Suggested terms
		-26.0	*working on the louder material*
		0.01	*catching only the peaks*
	attack_ms	6.5	*fast attack, grabbing transients*
		20.01	*moderate attack, letting transients through*
	release_ms	58.3	*quick release, lively, pumping*
		200.01	*slow release, smooth, unobtrusive*
limiter	threshold_db	-10.0	*clamped hard, well into the signal*
		-8.0	*holding down the louder peaks*
		0.01	*just shaving the peaks*
	release_ms	94.9	*quick release, letting go fast*
		200.01	*slow release, holding the level down*
gain	gain_db (default)	4.0	*a modest amount, noticeably*
		6.0	*substantially, a good deal*
		8.01	*drastically, way up or way down*
	gain_db (tuck_harmony)	2.5	*tucked just under the lead, close behind*
		4.5	*tucked clearly under the lead*
		6.01	*tucked well under the lead, kept in the background*
Shared frequency regions (bands_ref: shared.frequency_regions)			
	cutoff/region	100.0	*subsonic, infrasonic, below the audible low end*
		600.0	*low end, bass, body, boomy*
		1200.0	*low mids, boxiness, lower midrange*
		3400.0	*upper mids, presence, forward midrange*
		4800.0	*sibilance, harshness, bite, sharp*
		22000.0	*top end, air, brightness, crisp*
	Used by: highpass_filter.cutoff_frequency_hz; lowpass_filter.cutoff_frequency_hz; highshelf_filter.cutoff_frequency_hz; lowshelf_filter.cutoff_frequency_hz; peak_filter.cutoff_frequency_hz; introduce_low_frequency_rumble.{highpass_hz, lowpass_hz}		
highshelf filter	gain_db	6.2	*a gentle tilt, a slight shelf*
		7.4	*a moderate tilt*
		9.4	*a strong tilt*
		18.01	*a drastic tilt, the top end wholly rebalanced*
	q – pinned to 0.707; ignored		
lowshelf filter	gain_db	2.6	*a slight tilt, a nudge*
		5.5	*a moderate tilt*
		9.5	*a strong tilt*
		60.01	*a drastic tilt, the low end wholly rebuilt*
	q	0.83	*a gentle slope, a broad transition*
		1.0	*a standard slope*

continued on next page

Table 7 – continued from previous page

Effect	Parameter	Band upper	Suggested terms
		50.01	*a steep slope, a resonant turnover*
peak filter	gain_db	4.0	*a narrow trim, a light dip*
		12.01	*a firm notch, a decisive cut*
	q	3.16	*moderately narrow, focused*
		12.01	*narrow, surgical, pinpoint*
harmony	semitones	4.0	*a close harmony, a third*
		6.0	*a fourth or fifth apart*
		12.01	*a wide harmony, a fifth or more*
	key, mode – inherited from clip analysis; ignored		
mains hum	level_db	-27.0	*a faint hum, a low buzz*
		0.01	*an audible hum, an intrusive buzz*
	modulation_hz	0.265	*steady, unwavering*
		1.01	*slowly wavering, drifting in level*
	fundamental_hz – regional mains frequency; ignored		
low frequency rumble	level_db	-21.0	*a low-level rumble, a faint floor*
		0.01	*a heavy rumble, an intrusive low roar*
	modulation_hz	0.265	*steady, unwavering*
		1.01	*slowly wavering, breathing*
	highpass_hz, lowpass_hz – see shared frequency regions above		
Block bands (block_bands: send_return)			
send/return	send_level	0.4	*a light feed into the effect*
		0.65	*a moderate feed into the effect*
		0.9	*a strong feed into the effect*
		1.01	*the whole signal fed into the effect*
	return_level	0.55	*tucked well under the dry signal, subtle*
		0.725	*sitting under the dry signal, supportive*
		1.01	*clearly audible, level with the dry signal*
		1.2	*pushed above the dry signal, dominant*
	dry_level – pinned to 1.0 by every recipe; ignored		

D LLM JUDGE FRAMEWORK

D.1 PROMPTS

System Prompt

```
You are auditing one instruction written for a music-editing dataset.
You are given the symbolic edit graph it describes, the instruction one
level of abstraction below it, the rules that instruction was written
under, and what this level is licensed to discard. You decide two things
and nothing else. First, whether the instruction obeys its own rules.
Second, and separately, whether it correctly describes the instruction
it was derived from -- the same edit, at less precision, still
truthfully. You are indifferent to style. You do not rewrite anything.
```

Rules

1. Judge the instruction only against the rules and contract supplied. Do not import preferences of your own about how it should read.
2. The contract says what this level MAY discard. Discarding something the contract permits is never an inconsistency, however much detail is lost. Adding processing the source does not describe always is.
3. For a banded level, a descriptor is correct when it lands in the band the graph specifies. The suggested wording is a suggestion: different words in the right band are correct, and the suggested words in the wrong band are not.
4. For consistency, judge fidelity to the source, not conformance to a contract. Ask only whether a reader of the rewrite would come away with a true picture of the edit the source describes. Classify each way it fails:
   - wrong_interpretation { the element IS in the source but the rewrite describes it incorrectly: a cut described as a boost, a large space described as subtle, the wrong stem, one processor mistaken for another. The information is present and wrong. Calling a reverb a delay is this, not hallucination.
   - hallucination { the rewrite describes processing, a parameter or a target that is not in the source at all. Something is ADDED while the rest stays intact.
   - omission { something the level was obliged to keep is missing. Being less detailed than the source is NOT omission where the licence above permits it; that is the abstraction working as intended.
5. A rewrite can fail in more than one way, so list every kind that applies. An empty list means it describes the source correctly.
6. Do not report the same defect under both verdicts. al_rules grades the text against its own level's rubric; consistency grades it against the source. A missing operator can be either or both, but say so once under each only if it genuinely fails both tests.
7. Cite rule failures by the index of the rubric item. An unattributable failure is not a failure.
8. Mechanical checks have already run separately. Judge only what they cannot settle.
9. Give each failing verdict exactly one sentence of justification, naming the rubric item or contract term it breaks. The two verdicts get their own sentence each: they answer different questions and can disagree, so one sentence covering both explains neither. A passing verdict needs none.

D.2 Rejection Reasons

For the evaluation set, the LLM judge rejected $7908/25000$ of the subsampled plans. Table 8 provides a breakdown of these rejections and the corresponding rejection categories provided by the judge.

Rejection reason	Count
[stylistic]	3353
[al_consistency]	1717
[al_consistency, stylistic]	1205
[al_consistency, al_rules]	912
[al_consistency, al_rules, stylistic]	703
[al_rules]	13
[al_rules, stylistic]	5

Table 8: Evaluation set rejection reasons & counts

For the train set, the LLM judge rejected $4856/25000$ of the subsampled plans. Table 9 provides a breakdown of these rejections and the corresponding rejection categories provided by the judge.

Note: since we are not using any captions or real ground-truth data for SFT plans, `stylistic` was not a criterion for rejection. The criterion `implementation_leak` is a regex-based rule that ensures exact MCP tool calls such as `separate_audio` or `apply_chorus_effect` do not make it into the instructions verbatim.

Rejection reason	Count
`[al_consistency]`	2995
`[al_consistency, al_rules]`	1844
`[al_rules]`	15
`[al_consistency, al_rules, implementation_leak]`	2

Table 9: SFT Training set rejection reasons & counts

E Agent Framework Details

E.1 Agent States

We keep track of four main states in the agent:

1. Working Audio State - The agent keeps track of the working audio, the active isolated source, the path of the stem, the path of the residual, and the latest version of the processed stem.
2. Planning/Assignment State - The tools selected during the planning step are stored in a list of 'assignments.' Each assignment tracks status (pending, in progress, retrying, done, failed) as well as the next pending assignment.
3. Execution State - Tracks all aspects of a tool-calling step including current assignment, active source, if a branch (see Appendix E.2) is open, if there are any remaining assignments, and if the audio is ready to return
4. Failure/Repair State - Tracks the failed tool name, failed tool assignment, and failed arguments to the tool call.

E.2 Enforcing Agent Behavior

We place a few mechanisms to steer the behavior of the agent.

- Separation and Mixing "Branches" - When a stem is isolated, a branch is opened and kept track of in the execution state and closed when the source is mixed back in with the residual. This branch keeps track of which source is isolated and the location of the corresponding residual track. The system will not allow another separation tool call or operation on another source until the source has been mixed in with the residual.
- Forced Separation - The system can force separation if the agent tries to operate on a source before isolating that source
- Branch Recombination Gate - If a branch is open and there is no pending assignment, the system can force a mix tool call so the agent does not stray
- Forced Pending Assignment Processing - If the agent tries to return or mix sources before it has completed all of its assignments, the system will force it to perform another tool call. Forced return - If there are no more assignments and no branch is open, the system will force the agent to return the audio

E.3 Instrument Priors

We show the distributions of individual instruments, as well as the number of unique instrument combinations (for only RIME-compatible stems) seen in the two reference datasets used to derive priors for the SFT training set in Table 10 .

Source	N	Vocals	Drums	Bass	Guitar	# unique combinations
Nunes & Ordanini (2014)	2,399	96%	96%	88%	39–79%	not reported
MoisesDB (Pereira et al., 2023)	240	99.6%	99.2%	98.3%	92.5%	5/15

Table 10: Instrument distribution by source. For Nunes & Ordanini (2014), guitar representation is given as a range (39–79%) because separate statistics are reported for "distorted guitar", "acoustic guitar", and "clean guitar", and their overlap is not reported.

F RIME VALIDATION INTERFACES

We introduce three user interfaces for interactive validation: an edit graph plan generation/subsampling interface, an instruction generation interface, and an interface to test different parameter ranges for RIME effects. Instructions to run both interfaces are included in the RIME repository.

F.1 EDIT GRAPH UI

We include an interface to construct edit graphs from input tracks (Figure 14). From tracks, we can generate edit graphs using RIME recipes, motifs, distribution priors, and constraints. Those edit graphs can then be applied to the source audio and listened back to validate the edits are realistic.

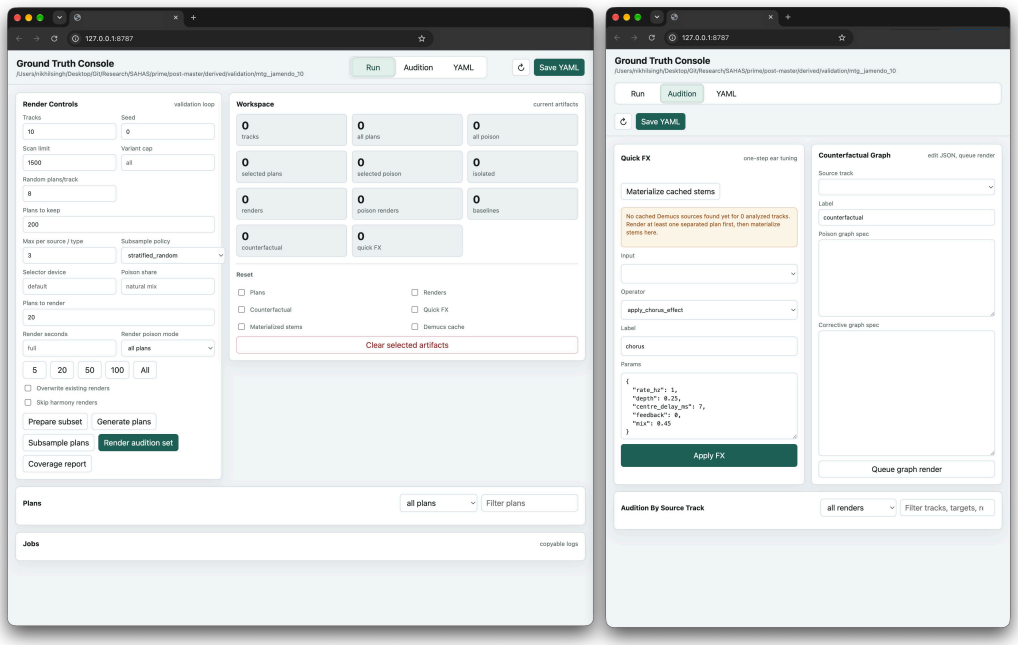


Figure 14: User interface for edit graph generation/subsampling. This was used to validate the procedure outlined in Algorithm 1 before generating plans at scale

F.2 INSTRUCTION GENERATION UI

The instruction generation UI (shown in Figure 15) allows you to build an edit graph and specify parameters for each effect and see the resulting instructions generated at different abstraction levels. We use this interface to test different instruction generation models and to assist with prompt engineering.

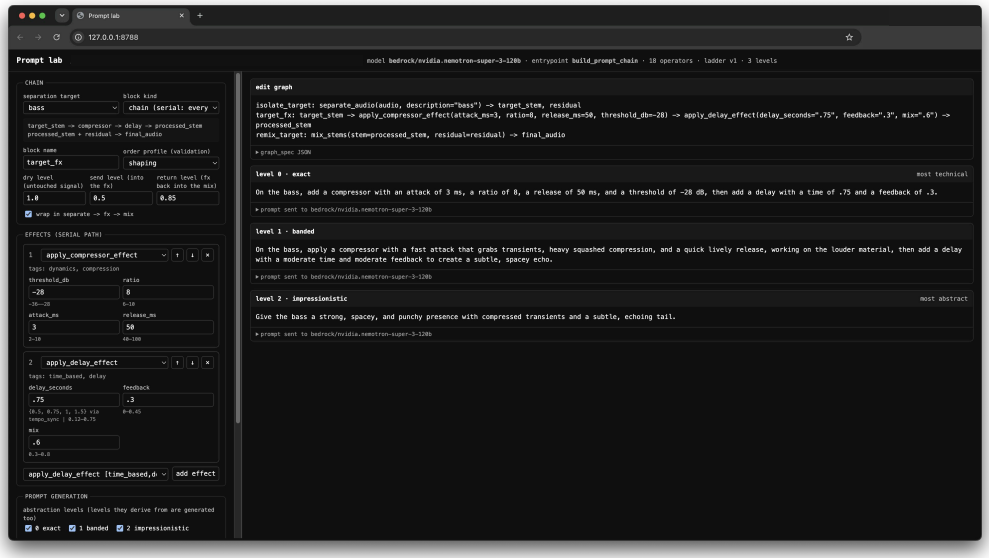


Figure 15: UI for instruction generation. Through this interface, we tested edit graph → instruction matching for various language models.

F.3 Parameter Testing UI

Figure 16 shows the parameter testing UI, which lets a user select a specific recipe/effect and sample different values from that effect's priors. A user can then apply that effect with the selected parameters to any input audio source. We used this UI to validate that the derived priors in Section 3.2 are realistic.

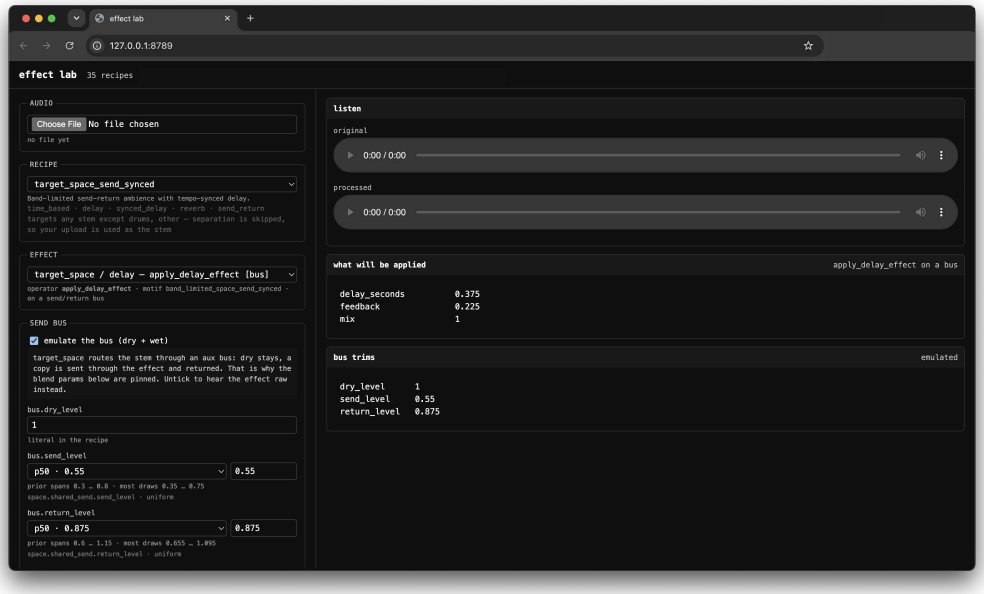


Figure 16: UI for parameter sampling, used to validate the priors outlined in Section 3.2 and Appendix B.4

G AUXILIARY RESULTS

G.1 REFERENCE AUDIO SIMILARITIES

Tables 11 and 12 report mean per-example cosine similarities using pooled MERT and CLAP embeddings, respectively. Input–target similarity measures the size of the reference edit; input–output similarity measures how close the agent stays to the input. Neither reference quantity alone measures task success. Output–target similarity measures absolute audio correspondence, whereas Δsim_a compares the input-relative edit directions. Graph F1 is the mean of kind, stem, operator, and edge F1. All 15,000 attempts per model are included, with failed runs treated as no-ops. CLAP input–target values differ across runs, so cross-model CLAP comparisons require caution.

Table 11: Mean reference and output audio similarities using pooled MERT embeddings, together with Graph F1. N counts evaluation attempts.

Model	N	Input–Target	Input–Output	Output–Target ↑	Δsim_a ↑	Graph F1 ↑
GEMINI 3 FLASH	15000	0.972	0.971	0.971	0.528	0.634
GPT-AUDIO-MINI	15000	0.972	0.971	0.979	0.637	0.821
GEMMA 3N	15000	0.972	0.958	0.965	0.609	0.857
GEMMA 3N SFT	15000	0.972	0.962	0.971	0.615	0.858

Table 12: The same reference and output similarities using the CLAP embeddings supplied with each run. Graph F1 is unchanged because it does not use audio embeddings.

Model	N	Input–Target	Input–Output	Output–Target ↑	Δsim_a ↑	Graph F1 ↑
GEMINI 3 FLASH	15000	0.763	0.782	0.858	0.625	0.634
GPT-AUDIO-MINI	15000	0.860	0.856	0.901	0.633	0.821
GEMMA 3N	15000	0.860	0.823	0.872	0.632	0.857
GEMMA 3N SFT	15000	0.763	0.728	0.877	0.730	0.858

G.2 HEAD-TO-HEAD MODEL PERFORMANCE

We compare models on the same edit and abstraction level using MERT output–target cosine similarity (Figures 17 and 18). A win requires strictly higher similarity; ties count as wins for neither model. GPT-AUDIO-MINI has the highest aggregate win rate, 59.0%, averaged across the three opponents and abstraction levels. At AL0, GEMMA 3N SFT beats both GEMINI 3 FLASH and GEMMA 3N in pairwise comparisons and has an aggregate win rate of 54.2%. Its aggregate rate drops to 45.2% at AL1 and 36.2% at AL2, where it is the lowest of the four models.

G.3 MODEL PERFORMANCE BY GRAPH COMPLEXITY

Figure 19 shows an expanded view of Figure 5, stratified by each model. All models show performance degradation as edit graphs get more complex, although this is not monotonic. Generally, they perform worse on the longest edit graphs at higher abstraction levels (though this is not true for GEMINI 3 FLASH audio match). While fine-tuning seems to help recover more complex graphs at AL0, it seems to have limited effect on AL1 and AL2.

G.4 FURTHER ABSTRACTION LEVEL ANALYSIS

We further explore the role of abstraction level in agent performance. Figure 20 highlights a clear decrease in both audio match and direction match for all models except GEMINI 3 FLASH from AL0 to AL2. Figure 21 shows a decrease by abstraction layer in audio composite and graph composite scores for both GPT-AUDIO-MINI, GEMMA 3N, and GEMMA 3N SFT. GEMMA 3N SFT outperforms all models in audio and graph match at AL0 and is comparable at AL1, although it falls behind substantially at AL2. Figure 22 shows that all models perform well at low ϵ values ($< .01$) only at AL0, and have almost no audio within tolerance values lower than .01 at AL1 and AL2. GEMMA 3N

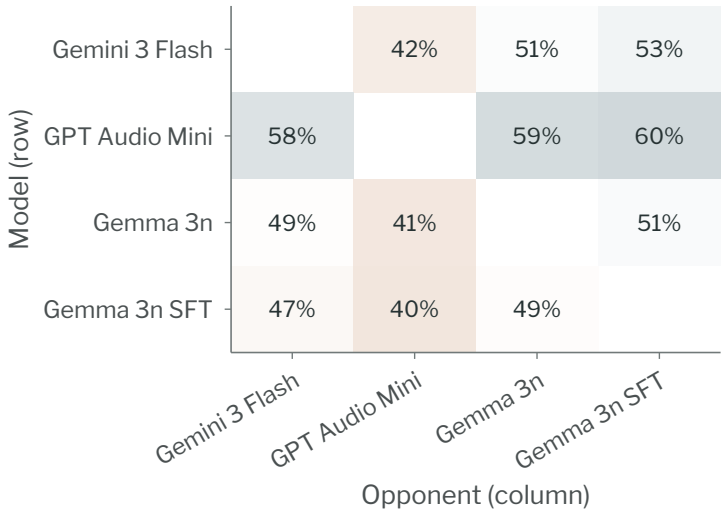


Figure 17: Model pairwise win rates for audio matching

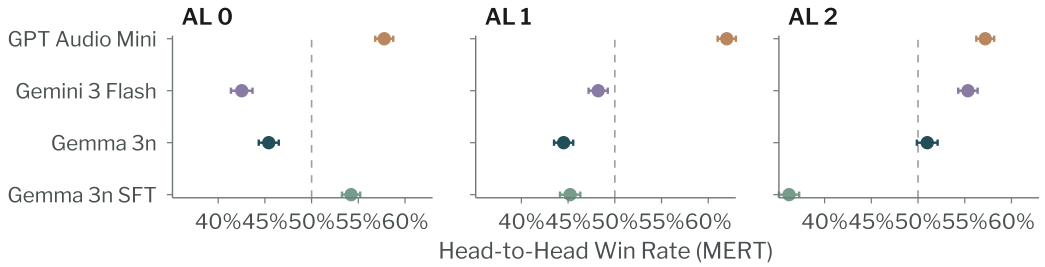


Figure 18: MERT audio-match win rates by abstraction level, averaged across opponents. Each model has 15,000 pairwise comparisons per level. Error bars are nominal 95% binomial intervals.

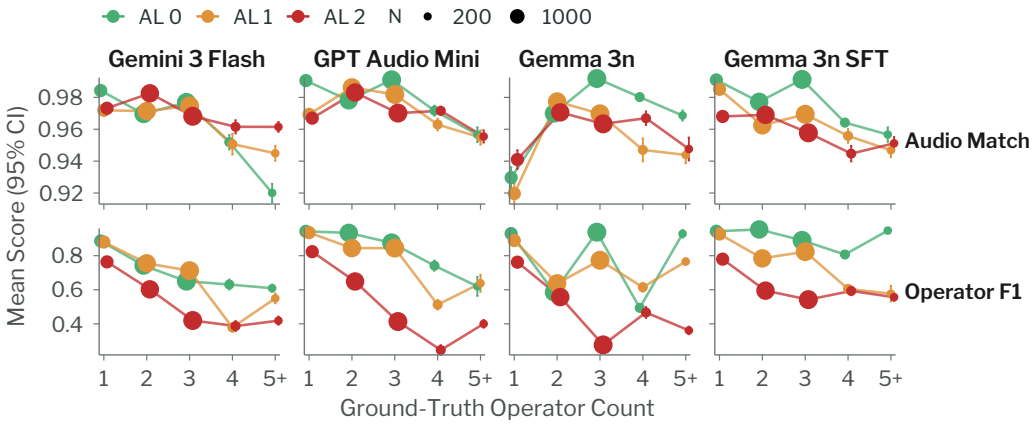


Figure 19: MERT output–target audio similarity and operator F1 by ground-truth operator count. Error bars show 95% CIs.

SFT indicates that fine-tuning is effective at both AL0 and AL1 (albeit to a lesser extent) but makes little difference at AL2.

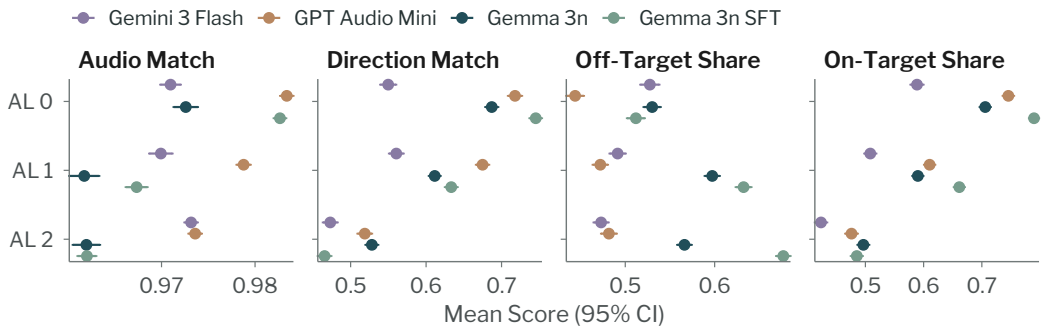


Figure 20: Audio and graph abstraction layer metrics

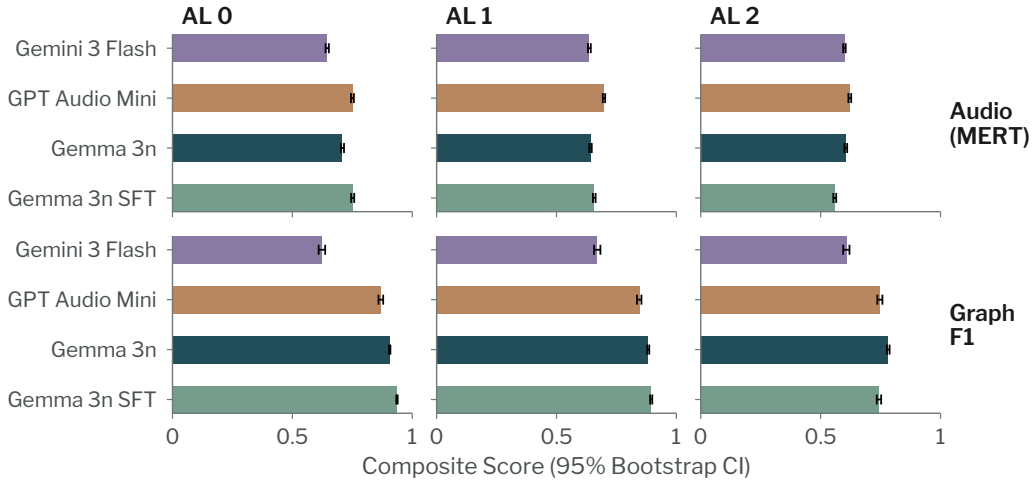


Figure 21: Audio and graph composite scores with confidence intervals by abstraction level

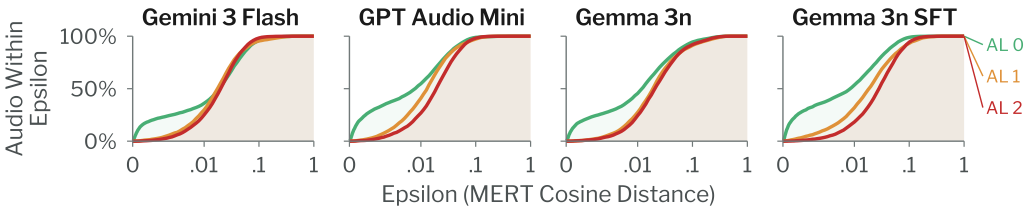


Figure 22: Epsilon thresholding accuracy by abstraction level

H Compute Resources

All Gemini 3 Flash and GPT-Audio-Mini experiments were performed via API calls to the respective provider APIs. Gemma 3n and Gemma 3n SFT models were hosted on Google Cloud and called via API. Gemma 3n SFT was trained on a H200, and embeddings were computed via a mixture of H200s and L40s.