Repente: A Specialized LLM for Musical Programming Languages with SonicUnit Knowledge Architecture

Architecture, Dataset Construction, Nine Iterative Training Cycles, and the Resolution of the Evaluation Battery

Carlos Eduardo Coelho Freire Batista

Centro de Informática · Laboratório de Aplicações de Vídeo Digital
Universidade Federal da Paraíba, João Pessoa, Brazil
bidu@lavid.ufpb.br

Preprint · August 2026

ABSTRACT

Repente is a local-first large language model specialized in musical programming languages. General-purpose code LLMs demonstrate broad competence across mainstream programming languages, yet they struggle with the domain-specific semantics of musical programming - DSP signal flow, temporal behavior, synthesis techniques, and the mapping between sonic intent and code implementation. This gap is addressed through fine-tuning Qwen2.5-Coder-7B-Instruct with QLoRA on a curated dataset built around the SonicUnit, defined as a reusable, parameterizable, and translatable unit of sonic behavior, organized along three representational layers (semantic, structural, perceptual) and five knowledge levels (syntax, DSP patterns, temporal, intent, cross-language).

The system evolved through nine iterative training cycles at approximately $284 total compute. The initial versions targeted three languages - SuperCollider, Pure Data, and MAX/MSP - over a 192,627-example corpus; from v0.3 onward the scope was deliberately narrowed to SuperCollider and Pure Data. Two models are released: v0.5, published first on the strength of a five-sample human evaluation, and v0.7, which a subsequent measurement establishes as the stronger of the two and which is therefore the current release. A baseline comparison against the unmodified Qwen model isolates what fine-tuning contributes, measured over 150 generations per model: the base model emits a well-formed canvas header in 2.7% of attempts against 100% for v0.5, and satisfies the full Pure Data criterion in 0.7% against 53.3% for v0.5 and 76.7% for v0.7. The base model names an output object more often than v0.5 does, so the deficit it exhibits is one of serialization rather than of intent. Fine-tuning also supplies SuperCollider edge cases and synesthetic vocabulary, at the documented cost of analysis depth compressed from 776 to 151 tokens through self-distillation.

The principal finding concerns the instrument rather than the models. Nine cycles were steered by a battery of five prompts sampled once each at temperature 0.7. Resampling that battery 30 times per prompt across all nine checkpoints shows that the ordering it produced is not the ordering of the models: v0.5, recorded at 5/5, has a measured expected score of 2.67 out of 5, and the probability of a model with those rates scoring 5/5 is 1.5%. Four of the five checkpoints after v0.4, v0.5 among them, are mutually indistinguishable at this sample size, so the ordering the battery reported within that

group was not an ordering at all. The v0.5i corpus increment, recorded as a regression from 5/5 to 2/5 and treated as evidence that the training set had reached a limit, measures 2.87, above v0.5 rather than below it, and the archived pair of scores is 2.8 times more likely with the labels reversed. Two draws from a single unchanged model differ by at least one point roughly 70% of the time. The trajectory that emerges under measurement is monotone from v0.4 to v0.7 and carries no plateau, which reverses the conclusion the project drew from the same checkpoints. A preliminary prompting study on frozen weights raises format validity from 75% to 100% and eliminates output truncation at no compute cost. The entire pipeline runs on a single rented GPU, at a per-cycle compute cost between $18 and $69.

1 Introduction

Musical programming languages such as SuperCollider [McCartney, 2002], Pure Data [Puckette, 1996], and MAX/MSP occupy a particular position among programming languages. Unlike general-purpose languages, they encode temporal behavior, perceptual quality, and aesthetic intent alongside computational logic. A SynthDef in SuperCollider is simultaneously a program, an instrument specification, and an implicit description of how something should sound. A Pd patch is at once a dataflow graph and a sonic architecture.

General-purpose code LLMs such as GitHub Copilot, Code Llama, and Qwen Coder have transformed software development productivity. Their training corpora, however, consist overwhelmingly of Python, JavaScript, Java, and other mainstream languages. Musical programming languages represent a small fraction of available training data, and the models' understanding of audio-domain semantics is correspondingly shallow. A typical code LLM generates syntactically plausible SuperCollider code yet often produces sonically meaningless results: oscillators connected to nothing, filters with parameters outside useful ranges, or timing structures bearing no relation to musical time.

At the other end of the spectrum, AI music systems such as Google's Magenta [Roberts et al., 2018], Meta's AudioCraft, and MusicGen [Copet et al., 2023] generate audio directly through waveforms, spectrograms, or MIDI sequences. Such systems bypass code entirely, offering no insight into how a sound is constructed, no ability to modify parameters, and no integration with the live coding and sound design workflows that musicians actually use.

Repente was conceived to occupy this gap: a model that treats musical programming languages as a domain rather than as syntax. The system targets two primary modes of operation - generation, understood as the translation of sonic intent into executable code, and analysis, understood as the explanation, debugging, and profiling of existing code. Later versions extend toward transformation, through style transfer and cross-language translation, and toward facilitation, through guided creative exploration.

The name Repente draws from the Brazilian tradition of the *repentistas*, improvisational poets who compose sung verse in real time under fixed metrical and rhyme schemes, so that invention occurs inside a form rather than in its absence. As with the repentista, the system aims at output that is at once structurally sound and responsive to the moment.

Methodologically, the working hypothesis at the outset of this project was that targeted dataset curation, rather than scale or architecture changes, would systematically resolve model weaknesses. Nine training cycles support that hypothesis, and the evaluation practice that accompanied them does not support the conclusions the project drew from it. Each cycle was assessed on a battery of five prompts sampled once each, and each decision about what to curate next, which version to publish, and when to stop was taken on that basis. Resampling the same battery across all nine checkpoints, reported in

Section 7.6, shows the resulting ordering to be largely an artifact of the sample size: the version selected for publication is indistinguishable from three of the four that followed it and measurably below the fourth, and a corpus increment recorded as a regression is in fact a small improvement.

The contribution of this paper is therefore threefold - a functioning specialized model, with two releases; an account of what domain fine-tuning supplies and at what cost; and a quantified demonstration that a five-sample battery cannot resolve the differences it was used to adjudicate, together with what the trajectory looks like once it is measured properly.

1.1 Contributions

1. **The SonicUnit knowledge architecture**, a formal representation of sonic behavior as the fundamental unit of knowledge for musical programming LLMs, described along three representational layers (semantic, structural, perceptual) and five knowledge levels (L1-L5, four of them implemented).

2. **A dataset construction pipeline** comprising configurable connectors, LLM-based enrichment, and multi-version validation, applied to collect and curate the training corpus across nine iterative versions.

3. **A baseline comparison quantifying the fine-tuning delta**: Pd format (+4/5), Pd connections (+4/5), SC edge cases (+2/8), synesthetic vocabulary (+100%), and, as a documented cost, analysis depth (−81%).

4. **A three-category Pure Data taxonomy** (standalone, abstraction, control) with PlugData compatibility, confirmed through human evaluation.

5. **A measurement of the evaluation battery itself**: 30 samples per prompt on each of nine checkpoints, 1,350 generations in total, yielding per-version rates with bootstrap intervals and an explicit resolution floor for the five-sample protocol.

6. **A corrected trajectory**: measured expected battery scores rising from 0.03 for the unmodified base to 3.83 for v0.7, monotone from v0.4 onward, with no plateau and with v0.7 statistically separated from v0.5. Three rules the project had adopted as absolute are shown to rest on single draws.

7. **A preliminary demonstration of prompt-level recovery**, in which few-shot exemplars combined with chain-of-thought prompting on the frozen v0.5 weights raise format validity from 75% to 100% and eliminate truncation at zero compute cost.

8. **A reproducibility-oriented, fully documented pipeline** at approximately $284 total compute across nine versions, including an account of the early v0.1 and v0.2 results that grounded every later decision.

2 Background and Related Work

2.1 Code-Oriented LLMs

Code generation models have evolved rapidly. Codex [Chen et al., 2021] demonstrated that language models could generate functional code from natural language descriptions. Subsequent models - StarCoder [Li et al., 2023], Code Llama [Rozière et al., 2024], DeepSeek-Coder [Guo et al., 2024], and Qwen2.5-Coder [Hui et al., 2024] - progressively improved code generation quality. Parameter-efficient fine-tuning methods, particularly QLoRA [Dettmers et al., 2023], have made domain adaptation accessible on consumer hardware.

These models are trained predominantly on high-resource languages, however, and struggle with low-resource, domain-specific languages for which training data is orders of magnitude smaller.

2.2 AI Systems for Music

AI-driven music systems have followed a different trajectory. Magenta [Roberts et al., 2018] explored neural network-based music generation via symbolic representations. AudioCraft and MusicGen [Copet et al., 2023] generate audio waveforms directly from text descriptions. RAVE [Caillon and Esling, 2021] enables real-time audio variational autoencoding. Such systems generate sound but not code; they do not produce editable, parameterizable, executable programs.

2.3 Prompt-Level Adaptation

Alongside weight-level adaptation, a body of work addresses capability elicitation without retraining. In-context learning [Brown et al., 2020] demonstrated that exemplars supplied at inference time steer model behavior, and chain-of-thought prompting [Wei et al., 2022] showed that eliciting intermediate reasoning improves performance on structured tasks. Such techniques are typically evaluated on general-purpose base models rather than on models already specialized through fine-tuning, and the interaction between a strong fine-tuned prior and in-context exemplars remains sparsely characterized. Section 8 reports a preliminary result on precisely that interaction.

2.4 The Gap

To the authors' knowledge, no existing system specifically targets the intersection of code LLMs and musical programming languages. The closest related work includes IDE-level autocomplete for SuperCollider and template-based code generation in MAX/MSP - rule-based systems with no semantic understanding of sonic intent, no cross-language reasoning, and no natural language interaction. Such approaches cover surface completion, yet leave uncovered the mapping between described sonic intent and executable implementation. Repente occupies this unexplored intersection: a model that reasons about code as sound.

3 The SonicUnit Knowledge Architecture

At the core of Repente's knowledge representation is the SonicUnit, defined as a reusable, parameterizable, and translatable unit of sonic behavior. The concept emerged from a foundational question: what is the atomic unit of knowledge that a musical programming LLM needs to learn?

It is not a line of code, nor a function, nor a complete program. It is a mapping between sonic intent, temporal behavior, and technical implementation, structured so that it can be understood semantically (described in natural language, including synesthetic vocabulary), executed computationally (rendered as valid code in any target language), and transformed across languages and contexts.

The SonicUnit is characterized along two complementary axes: three representational layers, which describe *how* sonic behavior is represented internally, and five knowledge levels, which describe *what* the model progressively learns across training versions.

3.1 Three Representational Layers

The semantic layer, through the interlingual translation analogy. Machine translation improved substantially when systems moved from direct language-pair translation to an intermediate representation. Repente likewise reasons through an intermediate semantic representation of sonic behavior, a musical interlingua.

The structural layer, through the SMILES notation analogy. In chemistry, SMILES notation serializes complex three-dimensional molecular structures into linear strings that machine learning models can process. An analogous compact notation for DSP signal graphs is envisioned - a serialization of the processing chain that captures structure independently of language syntax.

The perceptual layer, through the audio descriptor analogy. Future versions will incorporate audio descriptors such as spectral centroid, roughness, and brightness, extracted from actual code execution, enabling the model to map acoustic features to code. This layer is planned for a later multimodal version.

3.2 Five Knowledge Levels

The three layers are realized incrementally through five knowledge levels, four of which are implemented as of v0.5:

- **L1, syntax:** valid code, introduced in v0.1 and refined throughout.
- **L2, DSP patterns:** categorized audio processing, from v0.5.
- **L3, temporal:** scheduling and patterns, from v0.4.
- **L4, intent:** description-to-code mapping with synesthetic vocabulary, from v0.3.
- **L5, cross-language:** SC-to-Pd translation, with data generated in v0.6 and evaluation pending.

Because DSP concepts at L2 are largely universal while surface syntax differs per language, the architecture is naturally extensible: each new target language can be treated as an additional L5 skin over shared representations.

4 Method

4.1 Base Model and Fine-Tuning

Qwen2.5-Coder-7B-Instruct [Hui et al., 2024] was selected as the base model on account of its code-oriented pretraining and its permissive license. Adaptation was performed through QLoRA [Dettmers et al., 2023] at rank 32, yielding 80.7 million trainable parameters. The resulting adapters were merged and exported to GGUF with `Q4_K_M` quantization for local inference through `llama.cpp`, so that the deployed artifact runs on consumer hardware without a persistent GPU allocation.

4.2 Dataset Construction Pipeline

The corpus was assembled through a pipeline of configurable connectors targeting public repositories, documentation, and community patch archives, followed by normalization, deduplication, and LLM-assisted enrichment in which raw code artifacts were paired with natural-language descriptions carrying synesthetic vocabulary. A total of 27,935 examples were obtained through GitHub scraping. Pure Data material passed through an additional validation filter that reduced 18,625 candidate patches to 14,397 usable examples, distributed across the three-category system described in Section 5.3.

4.3 Version-Aware Validation

Each version was validated against a fixed battery of prompts covering generation and analysis in both target languages, with Pure Data output further subjected to human evaluation in PlugData, since format validity alone does not establish that a patch produces

sound. The validation protocol is described in Section 4.4, and version-by-version results appear in Section 7.

4.4 Evaluation Protocol

Evaluation combined automated and human components. Automated checks verified format validity, presence of connections, and absence of markdown wrapping. Human evaluation, designated Battery G for Pure Data and Battery F for SuperCollider composition, required an evaluator to load each generated artifact in its native environment - PlugData for Pd, the SuperCollider IDE for SC - to enable DSP, and to record whether the artifact produced sound, whether edits were required, and whether library-specific objects were used.

Human evaluation was conducted at $N = 5$ for Pure Data, with one sample drawn per prompt. This protocol was adopted because no automated substitute existed at the time, and it governed every subsequent decision about curation and release. Its resolution was never characterized. Section 7.6 characterizes it after the fact, by resampling the same five prompts 30 times on each of the nine checkpoints, and reports how much of the version ordering it produced survives that resampling. The archived figures are retained throughout this paper as the record of what the project observed, and are accompanied wherever they appear by the measured rate for the same model.

5 Dataset

5.1 Initial Composition (v0.1)

The v0.1 corpus comprised 192,627 examples spanning SuperCollider, Pure Data, and MAX/MSP, with an analysis-mode share of 18%. The remaining composition and the per-source breakdown are omitted here for length.

5.2 Dataset Evolution (v0.1 to v0.5)

Successive versions applied targeted corrections derived from the evaluation of the preceding version. In v0.2, analysis-mode examples were increased from 18% to 31% through LLM-assisted generation, and MAX/MSP sub-patcher text fields were preserved during normalization. In v0.3, MAX/MSP was removed entirely - the format had remained broken across two cycles, and the available corpus was too thin to support it - reducing the corpus to 67k examples concentrated on two languages. Versions v0.4 and v0.5 introduced automated correction passes, GitHub-scraped material, and the three-category Pure Data system, reaching 144,802 examples at v0.5, filtered down from 151,613 by removing examples above 6,000 characters.

The removal of MAX/MSP in v0.3 illustrates the general method: a capability measured as persistently deficient, with a diagnosable data-side cause, is either repaired through targeted data or removed from scope. Historical MAX/MSP results are retained in the appendices rather than discarded.

5.3 Three-Category Pure Data System

Pure Data material was partitioned into three categories - standalone patches carrying `dac~` or equivalent output objects, abstractions intended for instantiation inside a parent patch, and control patches operating on message rather than signal data. The partition matters because only standalone patches are directly verifiable through human evaluation: an abstraction loaded in isolation produces no sound by design. The filtered Pd corpus distributed as 1,714 standalone, 5,763 abstraction, and 6,920 control examples.

5.4 Known Limitations of the Initial Dataset

The initial corpus over-represented complex synthesis systems relative to minimal examples, a bias visible in the v0.1 evaluation, in which a request for a simple pad produced a sixteen-channel synthesis system. Library coverage was uneven, with ELSE objects sparsely represented throughout, a deficiency that persisted across every subsequent version and is analyzed in Section 9.4.

5.5 Post-v0.5 Corpora

Four corpora were constructed after v0.5, each targeting a deficiency identified through the five-sample battery. Version 0.6 added 3,016 Organelle patches, of which 99% were non-standalone, together with L5 cross-language pairs. Version 0.7 returned to the v0.5 base and added 5,000 standalone Pd patches with validated output objects plus 12,000 long-form analyses. Version 0.8 added thematic instrument examples with an ELSE-forcing strategy. Version 0.5i, constructed as a minimal-increment control, added approximately two percent additional data to the v0.5 corpus while leaving its composition otherwise intact. Their measured outcomes appear in Section 7.6.

6 Training

6.1 Configuration

Training used QLoRA over the base model quantized to 4-bit NF4, with LoRA rank 32 and alpha 64 applied across the attention and projection modules, yielding 80.7 million trainable parameters, approximately one percent of the total. The v0.1 run completed 6,020 steps. The v0.5 run used a sequence length of 1,024, per-device batch size 12, and gradient accumulation 2. Cycles v0.1 through v0.5 were trained on a single rented NVIDIA H100 SXM and cycles v0.5i through v0.8 on an H200 SXM; the change is visible in throughput, which rises from between 0.073 and 0.183 optimizer steps per second in the first group to between 0.220 and 0.237 in the second. Inference used GGUF `Q4_K_M` through `llama.cpp` with temperature 0.7, top-p 0.9, and a generation limit raised from 512 to 1,024 tokens from v0.4 onward, the earlier limit having truncated long analytical responses. Remaining hyperparameters appear in Appendix B.

6.2 Training Dynamics

Loss descended through four discernible phases across the v0.1 run, from 1.19 to 0.65, then to 0.42, and finally to 0.35, with final train loss of 0.4448 and train accuracy of 89.4%. Version 0.2, retrained with identical hyperparameters over the rebalanced corpus, reached a final-step train loss of 0.3559 and final-step train accuracy of 90.0%, with evaluation loss slightly higher at 0.4948 against a harder evaluation set. Its epoch-average train loss, at 0.4595, is above v0.1's 0.4448, so the two versions do not admit an unambiguous ordering on training loss.

6.3 Cost Analysis

Compute cost per cycle ranged from $18 to $69, with the higher figure corresponding to the larger v0.5 corpus. Cumulative compute across the nine cycles reached approximately $284, of which $158 was expended through v0.5, the first release, and $126 through the four subsequent cycles, which include v0.7, the current release. One-time LLM-assisted dataset enrichment added approximately $150 outside the per-cycle compute figure.

The $126 expended after v0.5 was recorded at the time as having produced no better model. Under measurement it produced one the battery can separate from v0.5, namely v0.7, which is the current release.

Table 1. Compute cost per training cycle, with the archived single-draw battery score and the measured expected score of Section 7.6 alongside. The two columns order the versions differently.

Version	Compute	Cumulative	Archived Pd	Measured
v0.1	$22	$22	three languages	not exported
v0.2	$20	$42	analysis rebalanced	1.80
v0.3	$18	$60	MAX/MSP removed	1.00
v0.4	$29	$89	Pd 1/5	1.67
v0.5	**$69**	**$158**	**Pd 5/5 - first release**	**2.67**
v0.6	$31	$189	Pd 3/5	3.27
v0.7	**$35**	**$224**	**Pd 4/5 - current release**	**3.83**
v0.8	$29	$253	no record	2.90
v0.5i	$31	$284	Pd 2/5, minimal increment	2.87

Both v0.5 and v0.7 remain published, and the reason is worth stating rather than leaving to inference. Version 0.5 is the artifact that the experiments of Sections 8 and 7.2 were run on, that the archived evaluations of Section 7 describe, and that any prior reference to this project points at; withdrawing it would orphan that record and would remove from circulation the object this paper is largely about. Version 0.7 is the model recommended for use. The two releases therefore serve different purposes, one as the frozen reference for the results reported here and one as the current artifact, and neither supersedes the other in the sense of correcting an error, since the measurement does not place v0.5 below v0.5i, v0.6 or v0.8.

7 Evaluation

Evaluation proceeded in three phases. The early phase, reported in Section 7.1, established on v0.1 and v0.2 what the model could and could not do, and grounded every subsequent dataset decision. The middle phase, in Sections 7.2 through 7.5, added a baseline comparison, human evaluation in PlugData, SuperCollider composition testing, and cross-platform verification, refining some of the early conclusions. The late phase, in Section 7.6, turns the instrument on itself and resamples the battery across all nine checkpoints.

7.1 Preliminary Evaluation (v0.1 and v0.2)

7.1.1 Protocol

The model was evaluated through 10 hand-crafted prompts covering both operating modes, with five generation and five analysis prompts across the target languages. Inference was performed with 4-bit quantization, temperature 0.7, top-p 0.9, and a maximum of 512 new tokens. Complete prompts and responses for both v0.1 and v0.2 are reproduced in Appendix A.

7.1.2 Generation Results (v0.1)

- **Test 1, SuperCollider SynthDef (warm detuned pad).** A 422-token SynthDef with sixteen-channel multichannel expansion, FM modulation, LFO control, and SplayAz spatialization: structurally valid SC code, yet overengineered, since the prompt requested a simple pad. The result indicated that the model had learned advanced patterns while failing to calibrate complexity to prompt scope.

- **Test 2, SuperCollider Pattern.** A 110-token idiomatic Pattern using `Pbind`, `Pseq`, `Pbindf`, and `Ppar`: satisfactory, though built on a fixed `Pseq` rather than random selection, and without explicitly setting 120 BPM.
- **Test 3, Pure Data FM Synthesizer.** A 134-token minimal patch reducing to `loadbang` and `outlet`: failed, bearing no relation to FM synthesis, the weakest result in the evaluation.
- **Test 4, Pure Data Karplus-Strong.** A 442-token patch with `noise~`, delay structure, and `hip~` filtering: partial, in that the core Karplus-Strong components were present while connections required adjustment.
- **Test 5, MAX/MSP Detuned Pad.** A 512-token `.maxpat` JSON with correct maxclass values and every newobj box missing its text field: broken, in that the model had learned the JSON skeleton without the object content, a direct consequence of dataset deficiency.

7.1.3 Analysis Results (v0.1)

All five analysis prompts produced responses of 6 to 26 tokens, functioning as classification labels rather than explanatory text. The model returned identifiers such as `synthdef: acid` instead of analyzing signal flow, failed to detect a critical feedback loop with gain above 1.0, and in one case responded with a generation instruction rather than an analysis. The root cause was read at the time as the 18% analysis-mode share in training, with the few analysis examples formatted as short classifications. The baseline comparison in Section 7.2 substantially revised this interpretation.

7.1.4 v0.2 Corrections and Results

Three targeted corrections produced v0.2: analysis-mode examples were increased from 18% to 31% through LLM-assisted generation, MAX/MSP sub-patcher text fields were preserved during normalization, and the dataset was re-exported and retrained under identical hyperparameters.

Table 2. Training metrics, v0.1 against v0.2, under identical hyperparameters. Train loss is reported on both bases because the two disagree in sign: v0.2 is worse on the epoch average and better at the final logged step, and its run stopped at 93% of the epoch, so its final-step value is not a completed-epoch quantity.

Metric	v0.1	v0.2	Note
Train loss, epoch average	0.4448	0.4595	3% higher
Train loss, final step	0.3844	0.3559	7% lower
Eval loss, final	0.4791	0.4948	Slightly higher, harder eval set
Train accuracy, final step	89.4%	90.0%	Improved
Eval accuracy, final	87.1%	86.6%	Comparable
Steps completed	6,020	5,500 / 5,925	93% of epoch

Generation improved markedly. Test 1 produced the correct architecture, a SawPad SynthDef with ADSR and `Tdef` loop, rather than an overengineered sixteen-channel system; Test 2 produced a richer `Pdef` with kick and snare instruments at 120 BPM; Test 4 referenced Karplus-Strong by name. In analysis mode, Test 10 produced a complete 251-token analysis with functional summary, sonic description, signal-flow narrative, and

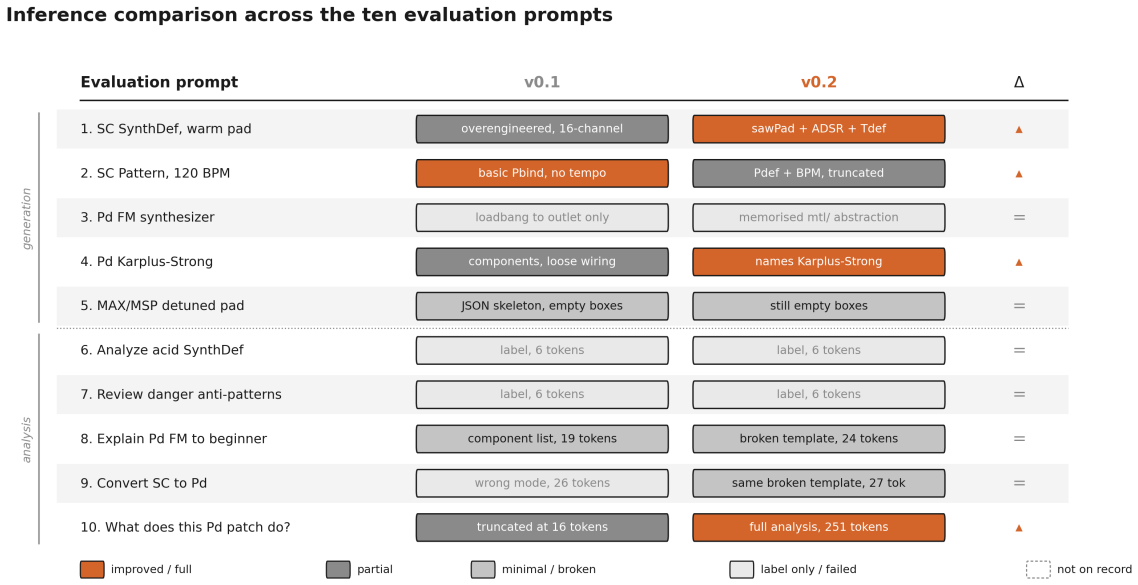


Figure 1. Inference comparison between v0.1 and v0.2 across the ten evaluation prompts, five in generation mode and five in analysis mode. Cell labels are transcribed from the per-test assessment records. Tests 3 and 8 carry no v0.2 assessment on record and are marked accordingly rather than inferred.

DSP categorization, matching the behavior the SonicUnit concept envisions. Analysis appeared prompt-dependent rather than model-limited, since the phrasing "Analyze this code" still collapsed to short labels while "What does this patch do?" triggered full analysis. MAX/MSP remained broken, motivating its removal in v0.3.

7.2 Baseline Comparison

Comparison against the unmodified Qwen2.5-Coder-7B sharpened, and in one respect overturned, the early conclusions. The base model already produces complete analytical responses, 6/6 and averaging 776 tokens, with step-by-step signal flow. Fine-tuning did not improve this and, as Section 9.2 discusses, compressed it.

What fine-tuning contributes is serialization. Measured over 150 generations per model, the base model emits a well-formed `#X connect` record in 2.0% of attempts and a valid `#N canvas` header in 2.7%, against 96.0% and 100% for v0.5. It nonetheless names an output object in 68.7% of attempts, which is more often than v0.5 does at 54.0%. The base model therefore knows that a Pd patch ends at `dac~` and does not know how to write the file that would carry it there, which locates the deficit precisely.

On the axes measured only through the archived single-draw protocol, and therefore subject to the resolution established in Section 7.6, fine-tuning also raised SuperCollider edge cases from 6/8 to 8/8, synesthetic vocabulary from 4 to 8 terms, and produced pure code output without markdown wrapping. The trade-off is analysis depth, compressed from 776 to 151 tokens through self-distillation across versions.

7.3 Pure Data Human Evaluation (Battery G)

Patches were tested in PlugData by a human evaluator. Version 0.5 produced sound in 5/5 patches, required zero edits in 4/5, used ELSE objects in 1/5, and produced correct connections in 5/5. This is the observation on which the decision to publish v0.5 was taken, and it is reported here as the project's record rather than as an estimate; Section 7.6 gives

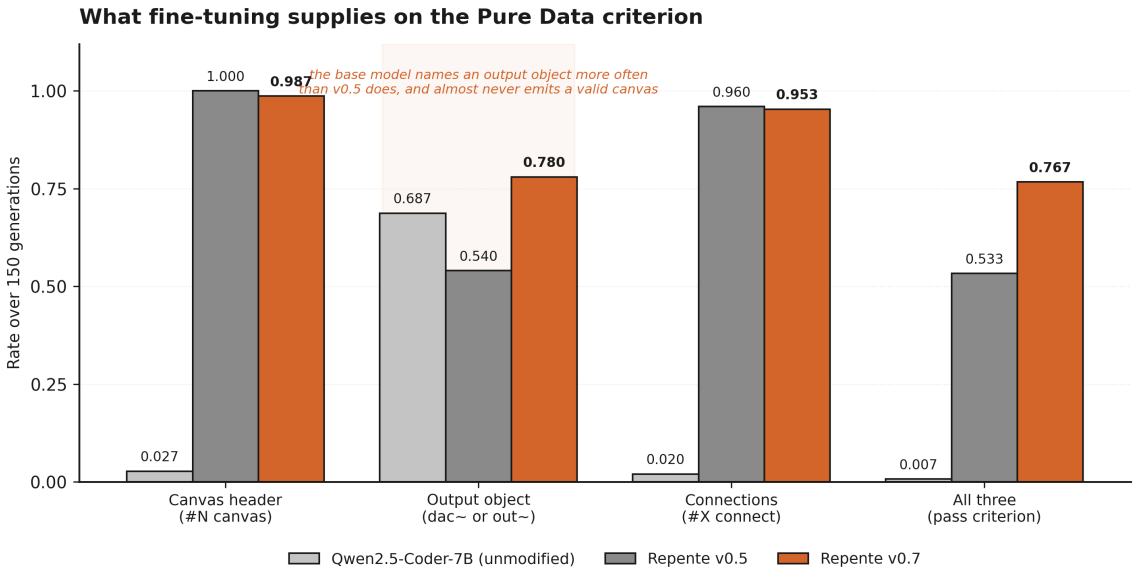


Figure 2. Components of the Pure Data criterion for the unmodified Qwen2.5-Coder-7B and for the two releases, each measured over 150 generations under the conditions of Section 7.6.

the measured rate for the same model.

The ELSE figure is the one archived result that measurement strengthens rather than overturns. Across the 1,350 generations of Section 7.6, an ELSE-prefixed object appears once, in v0.2, and never in any later version. The archived 1/5 for v0.5 was itself a favourable draw; the measured rate is zero. Five consecutive cycles of corpus weighting, including the explicit ELSE-forcing strategy of v0.8, left it there, which is the basis for treating library coverage as an inference-layer problem in Section 9.4.

7.4 SuperCollider Composition (Battery F)

Four-step iterative compositions were tested in the SuperCollider IDE. Compositions compile and play with limited editing, and became more compact across versions, which eliminated truncation. The principal remaining issue is that some compositions generate SynthDefs without accompanying playback code through `Pbind` or `Synth`, requiring the user to supply execution code.

7.5 Cross-Platform Consistency

Using GGUF `Q4_K_M` through `llama.cpp`, analysis matched 6/6 on Desktop with an RTX 5070 and on Mac with an M4 across five consecutive versions. Pure Data generation shows greater cross-platform variance, which suggests that Pd patch generation is more sensitive to the stochastic sampling process than either analysis or SC generation. Throughput measured 122 to 163 tokens per second on the desktop configuration and 12 to 14 tokens per second on the embedded target.

7.6 Resolution of the Evaluation Battery

The nine cycles were steered by the protocol of Section 4.4: five Pure Data prompts, one sample per prompt, temperature 0.7. That protocol was never characterized, and characterizing it is cheap, because it requires no retraining. Each of the nine checkpoints was resampled on the same five prompts, 30 times per prompt, at the archived conditions - temperature 0.7, top-p 0.9, a limit of 1,024 tokens, the same system prompt, distinct

decoding seeds - for 1,350 generations in total, on one machine, with no errors and no prompt returning identical output across seeds.

Scoring used the criterion already applied to the project's archived benchmark files: a generation passes when it carries a `#N canvas` header, an output object, and at least one `#X connect` record. This is weaker than the human criterion, since a patch may satisfy it and remain silent, and it is a necessary condition for the human criterion, since no patch produces sound without an output object and a connection to it. A measured pass rate therefore bounds the sound-producing rate from above.

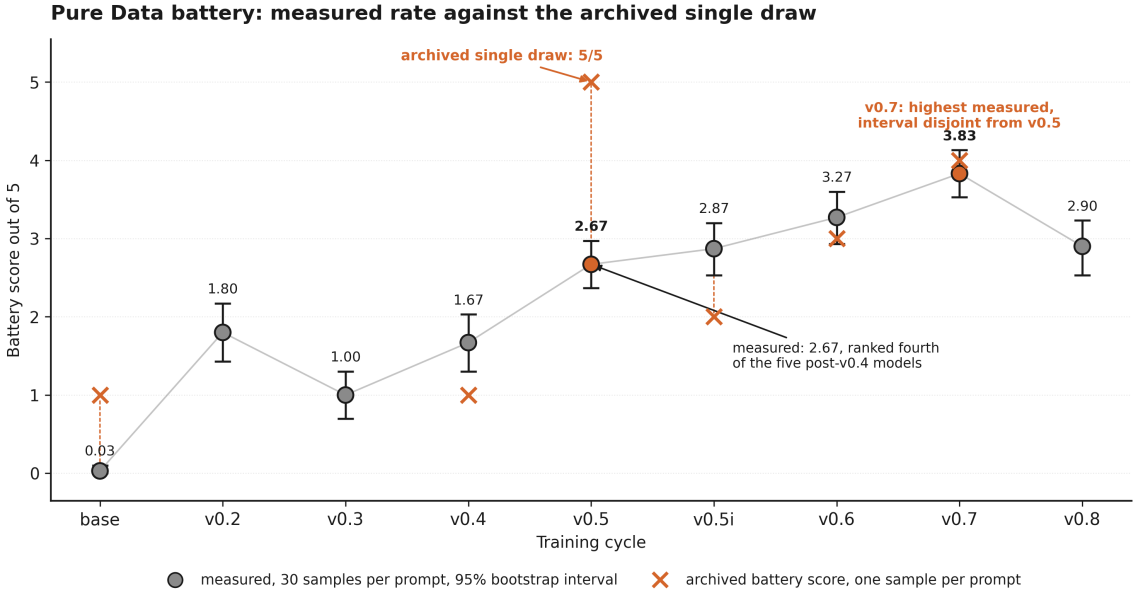


Figure 3. Expected battery score for each checkpoint, measured over 30 samples per prompt with a 95% bootstrap interval, against the archived score obtained from one sample per prompt. Dashed connectors mark the versions where the two disagree by more than 0.8 points.

Three results follow, and the first two concern the instrument rather than the models.

The archived scores are draws, not rates. Version 0.5 measures an expected 2.67 out of 5, with a 95% interval of [2.37, 2.97]. A model with those per-prompt rates scores 5/5 with probability 0.015. Since the criterion used here is necessary for producing sound, the archived observation of five patches producing sound had a probability of at most 1.5%, and the decision to publish v0.5 rested on it.

The battery cannot resolve the differences it was used to adjudicate. Two independent runs of the five-prompt protocol on a single unchanged model differ by at least one point with probability 0.67 for v0.5 and 0.70 for v0.5i, and by at least three points with probability 0.035 and 0.051. The v0.5 to v0.5i comparison, recorded as a fall from 5/5 to 2/5 and read as evidence that the corpus had reached a limit, is a gap of three. Under the measured rates the archived pair has probability 0.0042, while the same pair with the labels reversed has probability 0.0117, which is 2.8 times larger. The observation is not merely uninformative about which model is better; it points the wrong way.

The trajectory carries no plateau. Measured expected scores run 0.03 for the unmodified base, 1.80 at v0.2, 1.00 at v0.3, 1.67 at v0.4, 2.67 at v0.5, 2.87 at v0.5i, 3.27 at v0.6, 3.83 at v0.7, and 2.90 at v0.8. From v0.4 onward the sequence rises monotonically to v0.7 and falls only at v0.8. Version 0.7 carries the highest point estimate of the nine and is separated by

disjoint intervals from v0.5, v0.5i and v0.8; it is not separated from v0.6, whose interval overlaps it. It is accordingly released alongside v0.5 as the current model.

The separations the battery can and cannot make are worth stating explicitly, because reading the point estimates as a ranking is the error the rest of this section is about. Versions v0.5, v0.5i, v0.6 and v0.8 are mutually indistinguishable at this sample size, and v0.6 is also indistinguishable from v0.7. Of the ten pairwise comparisons among the five checkpoints following v0.4, only three are resolved, all of them involving v0.7. The band inside which the project believed it was observing regressions and recoveries contains four of the five candidates, and the archived scores for those four - 5/5, 2/5, 3/5, and no record - describe draws from that band rather than differences between models.

The dip at v0.3, which the archived record does not capture because Battery G had not yet been introduced, coincides with the removal of MAX/MSP and the reduction of the corpus to 67k examples. It recovers by v0.4 without intervention aimed at it.

Two limitations bound these results. The criterion is textual and does not establish that a patch produces sound, so the absolute levels reported here are upper bounds rather than sound rates; the ordering across versions is what the measurement supports, since the criterion was applied uniformly. And the measurement characterizes sampling variance at fixed weights, not the variance between training runs at fixed data, which would require retraining and is not addressed here.

8 Preliminary Study: Prompt-Level Recovery

This section reports a preliminary result. Its automated metrics are complete; its human evaluation is not, and its sample size is small. It is included because it bears directly on the conclusion of Section 7.6 and because its principal finding - a failure mode of few-shot prompting on fine-tuned models - is reproducible from the description given here.

8.1 Protocol

The v0.5 weights were held frozen. Four prompting conditions were evaluated over eight prompts arranged along a capability ladder: L1 prompts naming target objects explicitly, L2 prompts stating simple musical goals, L3 prompts using idiomatic musical language, and L4 prompts requiring compositional structure. The conditions were: **A**, the baseline system prompt; **B**, the baseline augmented with three worked exemplars pairing a musical request with a complete patch; **C**, an instruction to state one or two sentences of reasoning as a code comment before emitting the patch; and **D**, both exemplars and the reasoning instruction, with the exemplars themselves carrying the reasoning comment.

The primary automated metric was format validity, defined for Pure Data as the presence of a well-formed `#N canvas` header together with an output object, and for SuperCollider as the presence of executable playback structure. Truncation was recorded when generation reached the 1,024-token limit.

8.2 Results

Condition D achieves complete format validity across all four capability levels, eliminates truncation entirely, and does so while producing twelve percent fewer tokens than the baseline. The improvement over condition A is 25 percentage points of absolute format validity at zero compute cost, obtained without modifying a single weight.

8.3 Failure Modes

Two failure modes were identified, and both bear on how prompting interacts with a fine-tuned prior.

Table 3. Format validity by capability level and prompting condition, on the frozen v0.5 weights. Eight prompts per condition.

Condition	L1	L2	L3	L4	Mean	Truncated	Mean tokens
A, baseline	1.00	1.00	0.50	0.50	0.75	25%	415
B, few-shot	1.00	1.00	1.00	1.00	1.00	25%	536
C, chain-of-thought	0.00	1.00	0.50	0.00	0.38	12%	411
D, few-shot + CoT	1.00	1.00	1.00	1.00	**1.00**	**0%**	**366**

Context leak under few-shot alone. Condition B matches condition D on format validity while retaining the baseline truncation rate. Inspection of the truncated outputs shows the cause: after emitting a valid patch, the model continues the turn structure of the exemplars, hallucinating further user and assistant turns and regurgitating the supplied examples until the token limit is reached. In one L4 case the model copied the envelope skeleton from an exemplar and replicated it beyond one hundred times. The reasoning instruction in condition D suppresses this, apparently by supplying a plan-then-execute structure that terminates naturally.

Format degradation under chain-of-thought alone. Condition C reduces format validity to zero at L1, the level at which the baseline is perfect. Without exemplars to anchor the output shape, the reasoning instruction displaces the format prior established through fine-tuning: one L4 output under condition C produced an abstract timing circuit with no output object, structurally coherent and sonically inert.

The two findings taken together indicate that neither technique is independently safe on a fine-tuned model of this size. Exemplars supply format anchoring but invite continuation; reasoning instructions supply termination structure but erode format. The combination is what holds.

8.4 Limitations

The study covers eight prompts under four conditions, at a single temperature, on one model. Format validity is an automated proxy that establishes a patch is well-formed, not that it produces sound or that it satisfies the musical intent of the prompt; the human evaluation that would establish the latter is pending. The failure modes described above rest on inspection of individual outputs rather than on systematic coding. These results are therefore offered as a direction rather than as a measurement.

9 Discussion

9.1 Dataset Proportion versus Size

Version 0.6 added 5,480 non-standalone Pd examples against 136 standalone ones, a ratio of roughly 1:40 on the increment, and its archived battery score fell from 5/5 to 3/5. The inference drawn at the time was that proportion governs quality and that diluting the core capability degrades it. That inference guided the construction of v0.7, which restored the ratio.

The measurement does not support it. Version 0.6 measures 3.27 against 2.67 for v0.5, an improvement rather than a regression, and the archived fall from 5/5 to 3/5 lies well inside the band within which two runs of the battery on one unchanged model routinely differ. The 1:40 dilution did not degrade the capability it diluted.

What survives is narrower and still useful. Version 0.7 restored the standalone proportion and added 5,000 validated standalone patches, and it is the only checkpoint the

battery separates from v0.5. Proportion may well matter; the evidence assembled here does not isolate it, because v0.7 changed proportion and volume together, and the version that isolated a small change to the corpus, v0.5i, moved the score by less than the instrument can detect.

9.2 The Self-Distillation Problem, and the Evolving View of Analysis

Early evaluation interpreted analysis as a latent capability activated inconsistently through prompt phrasing. The baseline comparison refined this reading: the base model already produces complete 776-token analyses, so the capability was never absent, having been inherited from the base model. What varied was output length, and successive fine-tuning cycles drove it down.

The mechanism is self-distillation. Using the model's own outputs to train the next version creates a compression cycle, in which v0.2's short analyses of roughly 150 tokens trained v0.3 at roughly 140 tokens, which in turn trained v0.4, and so on. Analyses average 151 tokens despite the later addition of 12,000 long-form examples averaging 496 tokens, since those examples represent approximately 7% of all analysis examples, insufficient to overcome the roughly 50,000 short ones.

The two observations - early prompt-dependent latency and later self-distilled compression - are consistent rather than competing: the capability was always present, and what the prompt and the training mix modulated was how much of it surfaced. The prompt-side half of that statement is not settled by the evidence given here, since the early observation rests on informal comparison of two phrasings on a handful of prompts. Stated as a testable claim, it holds that the same model, given the same analytical task under systematically varied phrasings, produces responses whose coverage of the code under analysis differs beyond sampling variance. Measuring the between-phrasing variance against the within-phrasing variance would establish how much capability is recoverable at the prompt alone, and would bound what any future dataset intervention on analysis length needs to supply.

9.3 What Fine-Tuning Actually Does

The baseline comparison indicates that the value of fine-tuning for musical programming lies in format mastery and domain conventions rather than in general understanding. The Pd `#N canvas` format, with coordinate-based object placement and numbered connections, is unlike mainstream programming. General capabilities - analysis, code structure, explanation - transfer from the base model without fine-tuning.

The measurement of Section 7.6 separates that contribution into two components that arrive on very different schedules. Canvas serialization is acquired in a single cycle: the rate rises from 2.7% in the base model to 100% at v0.2 and stays there through v0.8. Output-object discipline, by contrast, improves across five cycles, from 33.3% at v0.4 to 78.0% at v0.7, and it is what the pass rate tracks almost exactly, since connection records are present in over 95% of generations from v0.2 onward. The first cycle taught the model to write the file. The remaining five taught it to remember where the file has to end.

Fine-tuning is not, however, the only mechanism that could supply surface form, and the contribution located above is therefore not yet decomposed. Grammar-constrained decoding enforces a serialization format deterministically, at negligible cost and without any training, and would guarantee that generated text is a sentence of the `.pd` format on any model, including one that had never seen the language. What such a mechanism cannot guarantee is referential consistency - that a connection record refers to object indices that exist and to ports compatible in type - because that property relates arbitrarily distant

records of the file and falls outside what a context-free grammar expresses, which places it in the class of constraints compilers defer to semantic analysis. The distinction suggests that the format mastery attributed to fine-tuning above may divide into a portion reproducible by deterministic masking and a portion that is not, and separating them requires a factorial design over grammar and fine-tuning that this work does not perform. Section 10 states it as a direction.

9.4 The Two Layers

Sections 7.6 and 8 read together suggest a division of labor between the two layers at which a specialized model can be improved.

Fine-tuning proved effective at installing what the base model did not possess: the Pd serialization format, its connection semantics, the convention of emitting code without prose. It kept improving the output-object discipline through v0.7, so the curation approach was not exhausted when the project stopped applying it.

Two deficiencies resisted it entirely. ELSE object coverage is zero across 1,350 measured generations in every version after v0.2, regardless of how the corpus was weighted and including a cycle that weighted it explicitly, and analysis depth remained compressed despite the addition of 12,000 long-form examples. Both are plausibly retrieval and elicitation problems rather than representation problems: the model does not lack the capacity to emit an ELSE object or a long analysis, it lacks, at inference time, either the documentation that would name the object or the prompt structure that would license the length. The prompting study supports this reading for the elicitation half, since format validity at L3 and L4 rose from 0.50 to 1.00 without any weight modification. The ELSE half is the stronger case: a capability that five corpus interventions could not move is unlikely to be a representation problem at all.

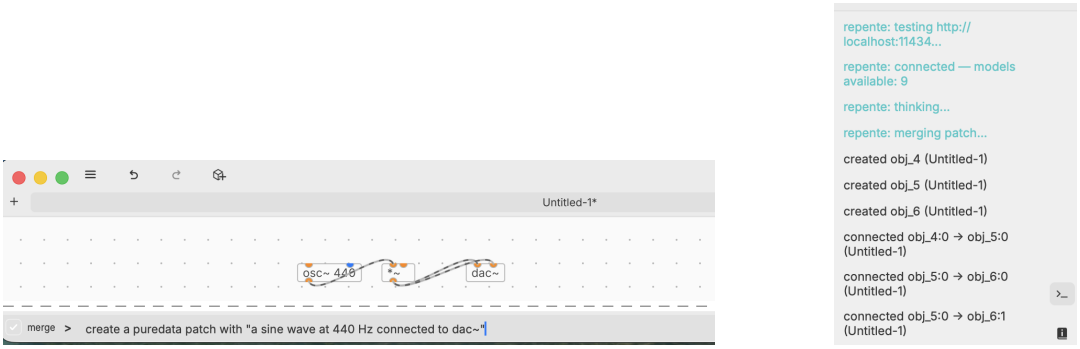
The practical consequence for a project operating at this budget is a sequencing rule rather than an abandonment of retraining. Deficiencies that a frozen model can be made to overcome through prompt structure or retrieved context should be probed there first, since the cost is zero and the iteration loop is minutes rather than hours, and the outcome tells the experimenter whether the capability is absent from the weights or merely unelicited. Retraining should then be directed at what survives that test. A second rule follows from Section 7.6 and is prior to the first: characterize the resolution of the instrument before letting it choose between candidates. The $126 spent after v0.5 did produce better models, and the project could not see them, which is a more expensive failure than spending the money would have been.

9.5 From Serialized Text to an Instantiated Patch

Every measurement in this paper scores text. The criterion of Section 7.6 asks whether a generation carries a canvas header, an output object and connection records, and the human battery of Section 4.4 asks an evaluator to open a file and listen. Neither observes the step in between, which is the one a musician performs: the serialized patch becoming objects on a live canvas.

Figure 4 shows that step. The request is typed into the patching window, and what reaches the canvas is not a file: the console records three `created` operations and three `connected` operations, so the model's output is parsed into canvas commands and applied incrementally. The distinction matters for failure behaviour. A malformed file fails at load and yields nothing; a malformed command sequence applies as far as it parses and leaves a partial patch, which is recoverable by hand and is the mode a user can work with.

The integration also carries the canvas state into each request as a system message, so



(a) prompt bar and canvas

(b) console trace

Figure 4. *pd-repente*, a fork of PlugData carrying a prompt bar in the patching window, serving the model over a local endpoint. The request in (a) produced three objects and three connections on the canvas directly, without a file being written or pasted; the console in (b) records the operations as they are applied. Panel (a) is a single screenshot with a region of empty canvas removed at the dashed rule; panel (b) is an unmodified crop of the same screenshot.

that a follow-up request is resolved against what is already there. That makes the unit of interaction an edit rather than a generation, which is closer to how patching actually proceeds and further still from what any of the measurements here score.

The patch in the figure makes the point sharper than intended. It satisfies the criterion of Section 7.6 completely: a canvas header, an output object, connection records whose indices all resolve, and both channels of `dac~` driven. It is also silent. The multiplier is instantiated as `*~` with no creation argument, which in Pure Data makes its right inlet a signal inlet defaulting to zero, and the console shows nothing connected to that inlet. The oscillator runs, the signal reaches the multiplier, and zero reaches the output.

This is the gap the paper has been describing, caught in a single figure. Every rate reported in Section 7.6 is an upper bound on the rate of patches that produce sound, and the distance between the two is populated by artifacts exactly like this one: structurally correct, referentially consistent, and inaudible. An instrument that loads the artifact and measures the signal, which Section 10 states as the precondition for everything that follows, would have caught it. The criterion used throughout this paper does not.

10 Future Work

Execution-grounded evaluation. The evaluation reported here decides whether a patch loads and produces sound by human listening, one sample at a time. An instrument that loads each artifact headlessly, renders audio to file, and decides the lower layers from deterministic descriptors of the signal would replace judgement with measurement on everything up to the point where musical adequacy begins, and would make every figure in Section 7 auditable by a third party from a provenance record. This is the precondition for the remaining directions rather than one item among them.

Musical benchmark design. Even instrumented, the evaluation above measures whether a patch produces sound, not whether it realizes the musical intent of the prompt. A request for a syncopated beat is satisfied by any patch that pulses. Until a benchmark exists that discriminates musical adequacy, further dataset investment proceeds without a signal to optimize against, repeating at larger scale the ambiguity that $N = 5$ introduced at small scale.

Capability ladder. The four levels used in Section 8 are proposed as an evaluation

scaffold: L1, explicit object naming; L2, simple musical goals; L3, idiomatic musical language; L4, compositional structure; extended by L5, cross-language translation, for which data was generated in v0.6 and evaluation remains pending. Version 0.5 is functional at L1 and L2, reaches L3 and L4 in format validity under condition D prompting, and is unmeasured at those levels for musical adequacy.

Curated musical corpus. The present corpus pairs technical prompts with patches. A corpus pairing idiomatic musical requests with idiomatic patches is the plausible route to L3 and L4 competence at the weight level. Given Section 7.6, any such corpus must be constructed as a controlled expansion of the v0.5 dataset with incremental evaluation at each step, rather than as a replacement.

Alternative adaptation strategies. Every cycle reported here used QLoRA at rank 32 over the same projections, with hyperparameters held near-constant so that dataset composition would be the only manipulated variable. That control was appropriate for the question then being asked and is precisely what leaves the adaptation method unexamined. Rank and target-module sweeps, full-precision LoRA, longer schedules at lower learning rate, and preference-based post-training over paired generations remain untried. Repeated runs at fixed data are the item of most immediate value: the measurement of Section 7.6 settles sampling variance at fixed weights and leaves training variance unaddressed, and three runs of one corpus with distinct seeds would bound it.

Alternative base models. Where the trajectory of Section 7.6 stops remains open, since it was still rising at v0.7 when curation ceased. Code-oriented bases of comparable or larger size differ in how much DSP and audio material their pretraining corpora contain, and that difference plausibly bounds what any downstream corpus can install. Evaluating candidate bases unmodified, at the sample size used here, would establish where the starting point sits.

Retrieval-augmented generation. Indexing SC, ELSE, Cyclone, and Pd vanilla help files addresses the ELSE coverage gap directly. A preliminary implementation over approximately 2,100 documents using dense retrieval was constructed and revealed a failure mode worth reporting: on queries naming several objects, retrieval covering only a subset of them degraded output more than retrieval supplied nothing, with the model abandoning Pd for SuperCollider. Partial coverage appears to perturb a strong fine-tuned prior more than absent coverage does. Object-aware retrieval, issuing one query per named object, is the indicated remedy and remains untested.

Grammar-constrained decoding. Constraining the sampler to a formal grammar of the `.pd` format guarantees surface form on any model without training. Applied factorially against fine-tuning, at graded levels of grammatical strictness, it would decompose the contribution located in Section 9.3 into the portion reproducible by deterministic masking and the portion that is not, and would establish whether a substantial part of what nine training cycles installed is available at inference for free.

Perceptual grounding. Executing generated code and analyzing the resulting audio through acoustic descriptors would close the loop between code and sound, realizing the perceptual layer of the SonicUnit and supplying the objective signal that the musical benchmark requires.

Embedded deployment. The integration shown in Section 9.5 runs on a desktop with the 7B model served locally. Distillation to a 3B model would put it on the Orange Pi 5 Plus, and speech input would remove the keyboard from the loop, at which point a spoken request becomes a running patch. Neither has been attempted.

11 Conclusion

Nine training cycles at approximately $284 total compute produced a model that generates valid SuperCollider and functional Pure Data patches, with consistent cross-platform behavior through GGUF quantization. Two versions are released: v0.5, published first, and v0.7, which measurement establishes as the stronger and which is the current release. The trajectory validates the iterative methodology the project set out to test, in which targeted dataset corrections resolve specific weaknesses: rebalancing analysis examples, narrowing the language scope through the removal of MAX/MSP, and introducing the three-category Pure Data system. Measured expected battery scores rise from 0.03 for the unmodified base to 3.83 at v0.7.

The evaluation practice does not survive the same scrutiny. Every one of those nine decisions was taken on five prompts sampled once each, a protocol whose resolution was never characterized. Characterizing it after the fact, by resampling the same five prompts thirty times on each checkpoint, shows that two runs on a single unchanged model differ by at least one point roughly seventy percent of the time; that the archived 5/5 which selected v0.5 for publication had a probability of at most 1.5%; that the fall from 5/5 to 2/5 read as evidence of a corpus limit is 2.8 times more likely with the labels reversed; and that the version chosen for release is indistinguishable from three of the four candidates that followed it, and measurably below v0.7, which the battery does separate. Three rules the project had adopted as absolute, concerning the immutability of the training set, the primacy of dataset proportion, and the exhaustion of curation, rest on single draws. The one archived finding that measurement strengthens is the negative one: ELSE objects appear once in 1,350 generations, so the library gap is real and is not a corpus problem.

Two conclusions follow. The first concerns what fine-tuning is for. Its contribution is serialization, and it divides in two: canvas format is acquired in one cycle and then flat, while the discipline of ending a patch at an output object improves across five, and it is the second that the pass rate tracks. General capabilities transfer from the base model without fine-tuning, and one of them, analysis depth, is actively degraded through self-distillation. The second concerns the order of operations. A preliminary prompting study on frozen weights raises format validity from 75% to 100% at no compute cost, which suggests probing the inference layer before retraining; but prior to either, an instrument has to be characterized before it is allowed to choose between candidates. The measurement reported here cost nothing beyond an afternoon of inference on hardware already owned, and it reversed the project's central conclusion.

More broadly, this work indicates that the gap between general-purpose code LLMs and domain-specific musical programming tools can be bridged with modest resources, and that the difficulty lies past the working model, in measuring it well enough to know whether the next cycle helped. The SonicUnit concept provides a framework for organizing knowledge about the relationship between sonic intent, temporal behavior, and code implementation, extending to additional languages, multimodal grounding, and cross-language reasoning. This document is released as a preprint to invite collaboration, feedback, and contributions from the computer music and machine learning communities.

Data and Code Availability

Everything reported here is released openly and collected at repente.net.

Models. Both releases are distributed as GGUF with `Q4_K_M` quantization from Hugging Face, under `bidubr`: `repente-v0.7-GGUF` is the current release, and `repente-v0.5-GGUF` the frozen reference on which the experiments of Sections 7.2 and 8 were run.

Measurement data. In the same repository, the 1,350 generations of Section 7.6, one record per generation with its seed, completion, token count, finish reason and per-criterion scoring, so that the reported rates can be recomputed rather than taken on trust. The archived per-version benchmark files that the historical scores derive from are included alongside them.

Code. At github.com/dobidu/repente: the generation and scoring harness, the analysis implementing the five rules of Appendix C, the scripts that produce Figures 1, 2 and 3 from those data files, and the LaTeX source of this document.

The integration. *pd-repente*, the fork of PlugData shown in Figure 4. It lives on the `pd-repente-main` branch of the repository at github.com/dobidu/plugdata.

Declaration of Generative AI Use

Generative language models appear in this work in three distinct roles, separated here because they carry different implications for how the results should be read.

As the object of study. The models evaluated throughout - Qwen2.5-Coder-7B-Instruct and the eight fine-tuned derivatives - are the subject of the paper rather than instruments used to write it. Their outputs are reported as data.

As a documented step of the method. The dataset construction pipeline of Section 4.2 used a commercial language model to pair raw code artifacts with natural-language descriptions carrying synesthetic vocabulary. This enrichment accounts for approximately $150 of the reported cost, is stated as a methodological choice rather than a writing aid, and its known consequence - the self-distillation of analysis length analysed in Section 9.2 - is reported as a finding rather than omitted.

As an authoring and analysis aid. The author used Anthropic's Claude models while preparing this manuscript: to draft and edit prose, to implement the figure-generation and measurement scripts, and to conduct the statistical analysis reported in Section 7.6 and Appendix C. Three safeguards apply to that use. The decision rules of the sampling-variance analysis were fixed and written down before any of its data existed, and are reproduced in Appendix C in the form in which they were declared. Every figure derives from a retained data file rather than from a description of one, and both the data files and the generating scripts are released as described above. Every quantity, claim and citation reported here was reviewed against its source by the author, who directed the work, took the interpretive decisions, and is solely responsible for the content, including any error that survived that review.

One consequence of this arrangement is worth stating plainly. The measurement of Section 7.6, which reverses a conclusion the project had held for four training cycles, was designed, executed and analysed in a single afternoon at no compute cost, and its being cheap is part of what the paper reports. The same tooling that made the analysis fast is capable of producing confident prose around numbers that were never checked, and the safeguards above exist because that failure mode is the one this paper is about.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 2020.

Antoine Caillon and Philippe Esling. RAVE: A variational autoencoder for fast and high-quality neural audio synthesis. *arXiv preprint arXiv:2111.05011*, 2021.

Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Jade Copet, Felix Kreuk, Itai Gat, et al. Simple and controllable music generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

Daya Guo, Qihao Zhu, Dejian Yang, et al. DeepSeek-Coder: When the large language model meets programming. *arXiv preprint arXiv:2401.14196*, 2024.

Binyuan Hui, Jian Yang, Zeyu Cui, et al. Qwen2.5-Coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Raymond Li, Loubna Ben Allal, Yangtian Zi, et al. StarCoder: May the source be with you! *arXiv preprint arXiv:2305.06161*, 2023.

James McCartney. Rethinking the computer music language: SuperCollider. *Computer Music Journal*, 26(4):61–68, 2002.

Miller Puckette. Pure data: Another integrated computer music environment. In *Proceedings of the Second Intercollege Computer Music Concerts*, pages 37–41, 1996.

Adam Roberts, Jesse Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, 2022.

A Complete Inference Results

This appendix reproduces the evaluation records for the ten prompts of the preliminary protocol described in Section 7.1. Sections A.1 through A.5 cover generation, Section A.6 covers analysis. Inference used the system prompt *"You are Repente, a musical programming expert specializing in SuperCollider, Pure Data, and MAX/MSP"*, temperature 0.7, top-p 0.9, and a limit of 512 new tokens, which is the limit that truncated Tests 5 and 10 and which was raised to 1,024 from v0.4 onward.

Response bodies are transcribed from the evaluation records. Where a record preserves only an excerpt, or where a body was not retained, the omission is marked rather than reconstructed.

A.1 Generation: SuperCollider SynthDef, warm detuned pad

Mode	Generation
Language	SuperCollider
Tokens / time	v0.1: 422 tokens / 18.5 s
Assessment	Overengineered. Valid structure, complexity far above the prompt's scope.

Prompt. *Write a SuperCollider SynthDef for a warm detuned pad with two sawtooth oscillators, a resonant lowpass filter with LFO modulation, and an ADSR envelope.*

v0.1 response (opening; full body not retained in the record):

```
SynthDef(\synth16, {
    arg out=0;
    var nch = 16;
    var amp=\levels.kr(1!nch);
    var f...
```

The response expanded to sixteen channels with FM modulation and SplayAz spatialization. It would compile and produce sound, though not the simple warm pad requested.

v0.2 response:

```
SynthDef(\sawPad, {|freq=100, amp=0.5,
  attackTime=0.3, decayTime=0.2,
  sustainLevel=0.7, releaseTime=0.4|
  var source;
  source = Saw.ar([freq, freq + 0.1],
    [amp, amp * 0.8]) *
    PanAz.ar(2, EnvGen.kr(
      Env.adsr(attackTime, decayTime,
        sustainLevel, releaseTime),
      doneAction: Done.freeSelf));
  Out.ar(0, source);
}).add;

~synth = Synth.new(\sawPad, []);
Tdef(\loopA, {
  var arr = [100, 200, 300];
  inf.do({ arr.do({|val|
    ~synth.set(\freq, val);
    1.wait;
  })});
}).play;
```

The v0.2 response matches the requested architecture: two detuned saw oscillators, an ADSR envelope, and a scheduling loop. The resonant lowpass filter with LFO modulation named in the prompt is absent, so prompt adherence remains partial even in the improved response.

A.2 Generation: SuperCollider Pattern, pentatonic melody

Mode	Generation
Language	SuperCollider
Tokens / time	v0.1: 110 tokens / 4.7 s
Assessment	Correct and idiomatic. Fixed sequence rather than random selection; tempo not set.

Prompt. *Create a SuperCollider Pbind pattern that plays a random melody on a pentatonic scale at 120 BPM.*

v0.1 response:

```
(
p = Pbind(\degree,
    Pseq([0, 2, 4, -1, -3], 5), \dur, 0.2);

q = Pbindf(p, \octave, 5);
r = Pbindf(p, \octave, 6);

x = Pseq([q,
    Pbindf(p, \octave, 5, \mtranspose, 12),
    r
]);

y = Ppar([x, x]);
y.play;
)
```

The degrees supplied form a pentatonic set and the pattern library is used idiomatically, so the failure is confined to two stated requirements: the melody is fixed rather than random, and 120 BPM is never set. In v0.2 the response became a richer `Pdef` deriving beat length from `beat = 60/bpm` with kick and snare instruments, which satisfies the tempo requirement, and it also exhibited a failure the paper meets again in Section 8: the model emitted three progressively refined variants of the same block and was cut off mid-statement at the 512-token limit. The improvement in content and the onset of repetition arrive together.

A.3 Generation: Pure Data FM synthesizer

Mode	Generation
Language	Pure Data
Tokens / time	v0.1: 134 tokens / 5.7 s
Assessment	Failed. A trivial `loadbang` to `outlet` patch with no audio content.

Prompt. *Write a Pure Data patch for a simple FM synthesizer with a sine carrier modulated by another sine oscillator.*

v0.1 response:

```
#N canvas 0 23 450 300;
#X obj 179 84 loadbang;
#X obj 178 169 outlet;
#X obj 180 110 t b;
#X msg 180 134 0;
#X connect 0 0 2 0;
#X connect 2 0 3 0;
#X connect 3 0 1 0;
#X coords 0 -1 1 1 15 15 1 0 800;
```

A `loadbang` that fires a bang, converts it to the float zero, and sends it to an `outlet`. There is no oscillator, no multiplication, no output object, and the trailing `#X coords` record marks it as a graph-on-parent abstraction rather than a patch. Nothing in it relates to frequency modulation.

v0.2 response:

```
#N canvas 1082 379 465 238 10;
#X obj 25 102 mtl/out~;
#X text 23 6 Description: Brightens up a stereo signal.;
#X obj 112 76 hsl 128 15 0 1 0 0 empty empty empty -2 -8 0 10
    -262144 -1 -1 0 1;
#X floatatom 112 96 5 0 0 0 - - -, f 5;
#X obj 25 46 mtl/brightness~ /id bright;
#X obj 25 72 mtl/in~;
#X text 22 22 tags: fx~;
#X connect 2 0 3 0;
#X connect 3 0 0 1;
#X connect 4 0 0 0;
#X connect 5 0 4 0;
#X connect 5 1 4 1;
#X coords 0 0 1 1 133 33 0;
```

The v0.2 response is a well-formed patch that does something else entirely. It instantiates the object `mtl/brightness~` between an inlet and an outlet from the same library, exposes a slider, and carries its own documentation: a description reading that it brightens a stereo signal, and a tag line. The prompt asked for frequency modulation.

This is verbatim reproduction of a training example, documentation included, and it is the clearest instance of memorization in the ten tests. The `mtl/` prefix identifies a third-party library that the project's own Pd validation filter blocks from v0.5 onward, so the observation and the remedy are both on record. The measurement of Section 7.6 confirms the remedy took effect: library-prefixed objects appear thirteen times across the 150 generations sampled from v0.2 and never again in any later version.

The two versions therefore fail differently rather than one improving on the other. Version 0.1 produced nothing; v0.2 produced the wrong thing, fluently and from memory.

A.4 Generation: Pure Data Karplus-Strong

Mode	Generation
Language	Pure Data
Tokens / time	v0.1: 442 tokens / 18.8 s
Assessment	Partial. Core components present, connections and parameters requiring adjustment.

Prompt. *Create a Pure Data patch that implements Karplus-Strong plucked string synthesis.*

v0.1 response (excerpt, as retained in the record):

```
#X floatatom 181 239 5 0 0 0 - - -, f 5;
#X text 254 239 Feedback amount (0 to 1).;
#X obj 176 330 *~;
#X text 345 314 This abstraction is used by
  [sampler~] for its echo functionality.;
#X obj 54 375 *~ 0.1;
#X obj 54 404 dac~;
#X obj 54 348 hip~ 5;
#X connect 0 0 11 0;
#X connect 0 0 7 0;
    ... (truncated in the record)
```

Two properties of this excerpt are worth separating. The first is the one the assessment records: noise excitation, a delay structure, and `hip~` filtering are present, which is the skeleton of Karplus-Strong, while the connections do not wire them into a working feedback loop. The second is visible only in the text boxes. The comment "This abstraction is used by `[sampler~]` for its echo functionality" describes a component of some other patch, and the patch is not an abstraction. The model reproduced documentation from a scraped training example alongside the code it was imitating, which is a form of memorization that generation metrics based on format validity do not detect. Test 4 in v0.2 improved, naming Karplus-Strong explicitly.

A.5 Generation: MAX/MSP detuned pad

Mode	Generation
Language	MAX/MSP
Tokens / time	v0.1: 512 tokens, truncated / 21.8 s
Assessment	Broken. Valid `.maxpat` JSON structure with every `newobj` box missing its `text` field.

Prompt. *I want something warm and slowly evolving in MAX/MSP. A detuned pad with chorus.*

v0.1 response (excerpt):

```
{ "patcher": { "boxes": [
  { "box": { "maxclass": "newobj" } },
  { "box": { "maxclass": "comment" } },
  { "box": { "maxclass": "ezdac~" } },
  { "box": { "maxclass": "panel" } },
  { "box": { "maxclass": "newobj" } },
  { "box": { "maxclass": "mc.live.gain~" } },
  { "box": { "maxclass": "playlist~" } },
    ... (truncated; 25 boxes total, all newobj boxes missing text)
```

The `text` field is what names the MSP object a `newobj` box instantiates, so a patch of twenty-five boxes without it is a layout with no signal processing. The model had learned the container schema and the inventory of interface classes while learning nothing about their contents, which traced

directly to training examples carrying the same deficiency. The condition persisted through v0.2 and motivated the removal of MAX/MSP in v0.3.

A.6 Analysis: Tests 6 through 10

All five analysis prompts produced responses of 6 to 26 tokens in v0.1, functioning as classification labels rather than explanatory text. Table 4 records the responses for both versions.

Table 4. Analysis-mode responses, v0.1 against v0.2. Token counts are for the response as emitted. Tests 6 and 7 are byte-identical across versions.

#	Lang	Prompt	v0.1	v0.2
6	SC	Analyze the `\acid` SynthDef (LFSaw + LFPulse to RLPF to ADSR). Describe what it sounds like.	`synthdef: acid` (6 tokens)	`synthdef: acid` (6 tokens)
7	SC	Review the `\danger` SynthDef for anti-patterns (feedback above 1.0).	`synthdef: danger` (6 tokens)	`synthdef: danger` (6 tokens)
8	Pd	Explain this FM Pd patch to a beginner.	Component list, no explanation (19 tokens)	Broken template with synesthetic terms (24 tokens)
9	SC/Pd	Convert this SuperCollider Saw + RLPF patch to Pure Data.	Began generating instead of converting (26 tokens)	The same broken template, and the wrong language named (27 tokens)
10	Pd	What does this Pd noise and filter patch do?	Correct identification, truncated (16 tokens)	Complete analysis (251 tokens)

Tests 8 and 9 in v0.2 share a structure that neither version of the paper previously recorded. Both emit "<*Language*>: *something* <*adjective*>, <*adjective*>, <*adjective*>. *Think* <*clause*>...." and stop, Test 8 at 24 tokens and Test 9 at 27. Synesthetic vocabulary has appeared, which the v0.2 corpus was rebalanced to encourage, and it has appeared inside a scaffold the model fills partially and abandons, with the word *Think* left exposed as an artifact of the template. Test 9 also names SuperCollider while being asked to convert to Pure Data. The compression documented in Section 9.2 is visible here in the act: the corpus supplied a shape, and the model learned the shape rather than the content it was meant to carry.

Test 7 is the most consequential failure of the five. The `\danger` SynthDef contains a feedback path with gain above 1.0, which diverges when run, and the model was asked to review it for anti-patterns. It returned the name of the SynthDef. A label is not merely a short analysis; it is an analysis that cannot be wrong, and a review that cannot be wrong cannot detect a divergent feedback loop either.

A.6.1 Test 10, v0.2 response in full

The following is the complete v0.2 response to Test 10, the first extended analysis the model produced and the behavior the SonicUnit concept was designed to elicit. It is reproduced in full because the paper's argument about analysis depth in Section 9.2 rests on what this format contains and on the fact that later versions stopped producing it at this length.

```
A simple bandpass filter noise generator using noise~ as the source signal
with a low-pass filter set at 800Hz and resonance of 20dB, combined with an
oscillator to create amplitude modulation.

Sonic output: Produces a filtered hissing sound with metallic overtones due
to the high-Q filtering. The added phase modulation creates subtle
vibrato-like effects in the frequency response, making it buzzy and resonant
rather than pure noise. The final result is warm and percussive with
metallic characteristics typical of bandpass-filtered white noise.

Signal flow: White noise enters through noise~ and passes through a bandpass
filter centered at 800Hz with 20dB resonance, shaping the initial spectrum.
This filtered signal is then multiplied by 0.3 for volume control before
being sent to both left and right channels via dac~. Simultaneously, a slow
triangle wave oscillator (phasor~) at 0.5Hz generates a slowly varying sine
wave that multiplies against a base frequency of 2000Hz and adds 200Hz
offset to modulate the bandpass center frequency, creating vibrato-like
tonal variation.

DSP category: subtractive
Complexity: intermediate
```

The response carries four components: a functional summary, a sonic description in synesthetic vocabulary, a signal-flow narrative traced to the output, and a categorization. It also contains two errors. The filter is described as low-pass in the first paragraph and as bandpass in the third, and the modulation source is called a triangle wave while the object named is `phasor ~`. The response is therefore the right shape with imprecise content, which is the distinction the evaluation protocol of Section 4.4 cannot draw and which motivates the perceptual grounding proposed in Section 10.

A.6.2 Prompt phrasing as the controlling variable

Tests 6 and 7 were phrased as instructions to analyze and review; both collapsed to six-token labels in both versions. Test 10 was phrased as a question about what the patch does; it produced 251 tokens. The prompts address comparable material at comparable difficulty, so the difference in output length is attributable to phrasing rather than to the code under analysis. This observation, formed here on five prompts and two phrasings, is what Section 9.2 restates as a claim requiring measurement of between-phrasing against within-phrasing variance, and what the prompting study of Section 8 tests on the generation side.

B Inference and Training Configuration

B.1 Adaptation

The adapter configuration is identical across all nine cycles. It was verified against the 39 `adapter_config.json` files retained in the checkpoint archive, and no field varies among them.

Base model	`Qwen/Qwen2.5-Coder-7B-Instruct`
Base precision	4-bit NF4
LoRA rank	32
LoRA alpha	64
LoRA dropout	0.05
Bias	none
Task type	`CAUSAL_LM`
Target modules	`q_proj`, `k_proj`, `v_proj`, `o_proj`, `gate_proj`, `up_proj`, `down_proj`
Trainable parameters	80,740,352 of 7,696,356,864, or 1.05%

The target-module set covers the attention projections and the MLP projections, not attention alone. The rank of 64 that an earlier internal roadmap proposed for v0.7 was never executed.

B.2 Optimization

The following are constant across all nine cycles. They were verified against the serialized training arguments retained for each run.

Learning rate	2×10^{-4}
Warmup	100 steps
Weight decay	0.01
Gradient clipping	max norm 1.0
Epochs	1
Step cap	none
Precision	bf16, fp16 disabled
Gradient checkpointing	enabled
Seed	42
Data seed	unset

The seed is worth noting. It is 42 in every run, which is the reason Section 10 lists repeated training at fixed data as outstanding: no cycle in this project ever varied it, so training variance has not been observed even incidentally.

B.3 Per-cycle variation

One quantity varied. Batch size and gradient accumulation were adjusted twice, and the effective batch is constant at 24 from v0.3 onward.

Version	Batch	Accum.	Effective	Steps	Runtime
v0.1	16	2	32	6,020	9.4 h
v0.2	16	2	32	5,500 of 5,925	—
v0.3	12	2	24	—	7.1 h
v0.4	8	3	24	4,960	11.4 h
v0.5	12	2	24	6,034	23.0 h
v0.5i	12	2	24	6,159	7.8 h
v0.6	12	2	24	6,559	7.8 h
v0.7	12	2	24	7,150	8.7 h
v0.8	12	2	24	6,276	7.4 h

Two entries need comment. Version 0.2 stopped at 93% of its epoch, which is why Table 2 reports its final-step training loss as not a completed-epoch quantity. And the v0.5 run took 23.0 hours against 7 to 11 for every other cycle, at a comparable step count; the cause is a small number of very long examples in that corpus, which the 6,000-character filter described in Section 5.2 was introduced to remove.

The throughput break between v0.5 at 0.073 optimizer steps per second and v0.5i at 0.220 is the hardware change recorded in Section 6.1, from H100 SXM to H200 SXM.

B.4 Training dynamics per cycle

Training loss is reported on two bases because they can disagree in sign, as they do for v0.2. The epoch average is the figure the trainer reports at the end of a run; the final step is the last logged value. Evaluation figures are the last recorded for each run. The held-out split changed with the corpus, so evaluation loss is not strictly comparable across cycles; it is tabulated because Section 7.6 relates it to the measured battery score.

Version	Loss, epoch avg.	Loss, final step	Eval loss	Eval acc.	Measured score
v0.1	0.4448	0.3844	0.4791	87.06%	not exported
v0.2	0.4595	0.3559	0.4948	86.59%	1.80
v0.3	0.6980	—	—	—	1.00
v0.4	0.6887	—	0.6069	83.98%	1.67
v0.5	0.6275	—	0.7684	80.54%	2.67
v0.5i	0.6186	—	0.7670	80.59%	2.87
v0.6	0.5930	—	0.4081	88.75%	3.27
v0.7	0.5091	—	0.2688	92.29%	3.83
v0.8	0.5060	—	0.2686	92.39%	2.90

Version 0.3 has no `trainer_state.json` in the archive, so its loss is taken from the run log and its evaluation figures are unavailable.

Across the seven versions carrying both an evaluation figure and a measured battery score, the Spearman rank correlation between evaluation accuracy and measured score is $+0.61$. The two agree on the extremes, placing v0.5 at the bottom and v0.7 at or near the top, and the single draw of the archived human battery is the measurement that disagrees with both. Version 0.8 is the one genuine dissociation: it has the lowest evaluation loss of the nine and the third measured score. A usable automated signal was therefore available throughout and was not used to arbitrate between candidates.

B.5 Inference

Export format	GGUF, `Q4_K_M` quantization
Serving	`llama.cpp`, OpenAI-compatible endpoint
Temperature	0.7
Top-p	0.9
Generation limit	512 tokens through v0.3, 1,024 from v0.4
Context window	4,096 tokens
Sequence length in training	1,024

The generation limit was raised at v0.4 because the earlier value truncated long analytical responses, a failure visible in Tests 5 and 10 of Appendix A.

Fields not tabulated here, including the optimizer and scheduler identifiers, are stored as enumeration references in the serialized training arguments and are recoverable from the archive files themselves, which are released as described in the availability statement.

C Sampling-Variance Protocol

This appendix records the protocol behind Section 7.6 in enough detail to reproduce it. The measurement required no retraining and no hardware beyond the machine on which the models were already served.

C.1 Design

The five Pure Data generation prompts of the archived battery were resampled 30 times each on nine checkpoints, giving 1,350 generations. The nine are the unmodified base model and the eight fine-tuned versions exported to GGUF; v0.1 has no GGUF export and is absent.

Every parameter was held at the value used in the archived measurement, since the object of the experiment is the archived protocol rather than an improved one:

Sampling temperature	0.7
Top-p	0.9
Generation limit	1,024 tokens
Context window	4,096 tokens
Quantization	GGUF `Q4_K_M`
Serving	`llama-server`, OpenAI-compatible endpoint
Decoding seeds	1000 to 1029, identical across models
System prompt	*You are Repente, a musical programming expert.*

Seeds were held identical across models so that the nine comparisons are paired. The server was restarted for each checkpoint and the loaded file name was read back from the endpoint and compared against the requested path before any generation was scored, since a label attached to the wrong checkpoint is the one error in this procedure that yields a plausible and false result.

C.2 Scoring criterion

A generation passes when it satisfies all three of:

- a `#N canvas` header is present;
- an output object is present, either `dac~` or `out~`;
- at least one `#X connect` record is present.

This is the criterion already applied to the project's archived benchmark files, adopted here so that the measured rates and the historical records are commensurable. It is weaker than the human criterion of Section 4.4, since a patch may satisfy it and remain silent, and it is necessary for that criterion, since no patch produces sound without an output object and a connection reaching it. Measured pass rates therefore bound sound-producing rates from above, which is what licenses the probability statement about the archived 5/5 in Section 7.6.

Recorded alongside the criterion, and not used in it: completion tokens, finish reason, distinct-output count per prompt, unique object count, and any object carrying a third-party library prefix.

C.3 Analysis, declared before the data existed

Five rules were fixed prior to execution.

R1. A per-prompt pass rate with a 95% Wilson score interval.

R2. The battery score treated as the sum of five independent Bernoulli draws at the estimated per-prompt rates, that is a Poisson-binomial, evaluated exactly.

R3. The probability of each archived score under *both* models of a compared pair. The cross terms decide the question: if the archived pair is comparably probable with the labels reversed, it carries no evidence about which model is stronger.

R4. The expected battery score with a 95% bootstrap interval over 20,000 resamples drawn within each prompt. Overlapping intervals are reported as indistinguishable on this battery at this sample size.

R5. The probability that two independent runs of the five-prompt protocol on a single unchanged model differ by at least g points, for g in 1, 2 and 3. This is the resolution floor of the protocol.

C.4 Execution record

All 1,350 generations completed with no request errors. No prompt returned identical output across its 30 seeds on any model, so the sampler varied throughout and the per-request seed was honoured; the highest repetition observed was two seeds coinciding on one output, which occurred in three of the forty-five prompt-model cells. Wall-clock time was between 6.5 and 17.2 minutes per model and 76 minutes in total, the base model being the slowest because its generations were the longest at 766 completion tokens on average against roughly 300 for the fine-tuned versions.

C.5 What the protocol does not settle

It characterizes sampling variance at fixed weights. Variance between training runs at fixed data is untouched and would require retraining, which Section 10 states as outstanding. It also does not measure sound: the criterion is textual, so the absolute levels are upper bounds, and it is the ordering across versions that the measurement supports, the criterion having been applied uniformly.