Executable but Wrong: Verifier-Grounded Contracts and Counterexamples for LLM-Generated Music Programs

engrXiv preprint — August 2026

ZHENGYANG DING, University of Electronic Science and Technology of China, China

A generated program can compile, execute, and still be wrong in the only way its user cares about. This is common in creative domain-specific languages: the artifact is partly subjective, yet requests often contain objective promises such as a tempo, onset grid, pitch range, or event count. Ordinary compilers establish executability, while self-critique leaves the model to invent its own oracle.

We present StruSpec, a verifier-grounded workflow for LLM-generated Strudel music programs. It turns explicit promises into bounded executable contracts, evaluates a deterministic Strudel subset with exact rational time and source provenance, and returns typed counterexamples to a preservation-aware repair loop. The model proposes patches; only re-execution and verification can end the loop. We validate the semantic boundary against the official runtime, an independently implemented semantics, generated oracles, metamorphic relations, controlled faults, and reviewed natural programs.

The evaluation separates four questions that are often conflated. Across 4,000 generations, 81.03% compiled but only 15.98% satisfied every hard constraint. On 600 paired failures, localized counterexamples improved strict repair success by 10.83 percentage points with GLM-5.2 (95% CI 8.17–13.67) and by 14.83 points in a protocol-matched DeepSeek V4 Pro replication (11.83–17.84). On a new 192-request, author-confirmed expert population, localized repair also beats generic feedback with both GLM (17 vs. 0 repairs) and DeepSeek (12 vs. 1), although only 67 requests reach a complete-trace semantic violation. A symbolic-first variant used 78.7% fewer model calls and 67.2% fewer output tokens, although its typical successful patch was no smaller. Conversely, supplying the same structured contract before any failure reduced first-attempt satisfaction: 23.17% for direct generation, 18.67% with a gold contract, and 17.54% with a parsed contract. The combined evidence reveals a phase asymmetry: structure is valuable when independent execution turns it into new diagnostic evidence, not when a redundant schema is merely placed in front of a generator.

CCS Concepts: • **Software and its engineering → Software testing and debugging**; *Software maintenance tools*; *Formal software verification*; Compilers.

Additional Key Words and Phrases: LLM-generated code, executable specification, automated program repair, counterexample-guided repair, domain-specific languages, music programming

1 Introduction

A music program can compile, run, and still miss the beat. The browser emits sound; the generated code looks idiomatic; there is no exception to chase. Yet the kick arrives an eighth-cycle late, the bass escapes its requested register, or one bar contains five events instead of four. These failures are interesting precisely because the surrounding artifact is creative. Whether a groove is compelling is a matter for a listener. Whether it has the promised tempo or onset grid is not.

The distinction exposes a weak point in the usual account of LLM code generation. Syntax-constrained decoding can keep output inside a grammar [29, 33]; compilation can reject malformed programs; tests can sample behavior; and a language model can be asked to criticize its own answer [7, 20]. None of these steps, by itself, establishes that an executable temporal program implements the user's explicit intent. This is an instance of the oracle problem [3], but with an additional difficulty: a compact pattern expression may create, move, or replicate events far from the source token that appears responsible.

Strudel makes this tension concrete. It brings the compositional pattern model of Tidal [21] into the browser [27]. A short expression can describe a long or even conceptually unbounded stream

Author's Contact Information: Zhengyang Ding, University of Electronic Science and Technology of China, Chengdu, China.

of musical events. Its terseness is excellent for live coding and difficult for generic repair feedback. Telling a model that "the program does not satisfy the prompt" asks it to rediscover the failed obligation, the time at which it fails, and the source derivation that caused the event. Telling it that a line compiled gives no semantic guidance at all.

This paper studies a deliberately bounded alternative. StruSpec places an executable contract between the request and the program. The contract records only explicit, checkable obligations; a conservative language boundary admits only deterministic Core Strudel; an instrumented evaluator produces exact finite event traces with provenance; and a three-valued checker emits either a proof of satisfaction, a localized counterexample, or an honest unknown. A repair model may propose candidates, but it cannot declare itself finished. The verifier re-executes every candidate and also checks that already satisfied obligations remain satisfied.

The core software-engineering question is not simply whether "more structure helps." Structure can play at least two different roles. Before generation, it is another representation that a model must translate into code. After a failure, independent execution can turn the same representation into evidence that was absent from the original request. We therefore ask:

RQ1: The semantic gap. How often does successful compilation hide violations of explicit musical requirements?

RQ2: Repair. Does verifier-produced localized feedback improve strict repair success over matched generic feedback, and does the effect replicate across repair-model families?

RQ3: Mechanism and cost. Which benefits come from symbolic repair, individual counterexample fields, or the complete iterative protocol?

RQ4: Phase. Does making the contract visible before generation improve first-attempt correctness, or is its value specific to diagnosis after failure?

The evidence supports a sharper story than a generic structure claim. In 4,000 crossed generations, 3,241 programs compiled, 2,561 produced complete Core traces, and only 639 satisfied every hard constraint. On a frozen set of 600 failures, localized counterexample-guided repair gained 10.83 percentage points over generic feedback with GLM-5.2 and 14.83 points in a matched DeepSeek replication. All claimed repairs were independently reverified. The gain did not reduce to two visually salient fields: in a token-balanced factorial study, minimal witnesses and source coordinates had near-zero isolated main effects. Nor did structure automatically improve generation. When the original controlled request already stated its obligations, supplying a correct gold MusicSpec reduced first-attempt satisfaction by 4.50 points.

We make three contributions, organized around authority, admission, and evidence:

- *Verifier authority:* a bounded executable contract for temporal programs, with exact event semantics, a three-valued oracle, provenance-bearing proof objects, and verifier-terminated, preservation-aware repair. We triangulate this authority against the official runtime, an independent semantics, generated reference cases, metamorphic properties, provenance ablations, and natural-program review.
- *Semantic admission:* StrudelSpecBench, spanning controlled, out-of-distribution, model-natural, and 332 expert-authored requests from 46 contributors, together with an explicit validation funnel. A prospective 192-request population shows that author-confirmed role binding separates interface failures from the semantic failures that repair can address.
- *Phase-dependent evidence:* paired and factorial studies across two repair-model families showing that executable intent improves repair when independent execution turns it into localized evidence, but that isolated fields do not explain the gain and a redundant pre-generation schema can impose a translation cost.

Our claim is intentionally narrower than "formal verification solves creative coding." StruSpec judges four explicit obligation families over finite traces of a deterministic language subset. It does not judge musical taste, and it does not turn unsupported programs into false assurances. That boundary is a feature of the design, not an omission to be hidden.

2 A Running Failure

Consider the request: *Create a two-track 120-BPM loop. Put a kick on every quarter-cycle and keep the bass between MIDI 36 and 48, with four bass events per cycle.* The following program is executable Strudel:

```
stack(
  s("bd*4").late(1/8).bank("RolandTR909").orbit(0),
  note("<c2 eb2 g2 bb2>").s("sawtooth").orbit(1)
)
```

It is also wrong. The `late(1/8)` transform shifts every kick away from the requested grid. A compiler can establish that the combinators exist and their arguments are well formed. Listening may reveal the displacement, but it does not produce a machine-checkable explanation, and the same approach does not scale to thousands of candidates.

2.1 The Contract

StruSpec extracts only the request's objective commitments. A simplified MusicSpec for the example appears below; the actual schema also carries stable identifiers, selectors, finite scopes, severity, and explicit clock evidence.

```
tempo:       120 quarter-notes/minute
onset_grid:  kick begin = 0 mod 1/4 cycle
pitch_range: 36 <= bass MIDI <= 48
event_count: exactly 4 bass onsets per cycle
scope:       cycles [0, 4)
```

The contract is not the composition. It leaves timbre, harmony within the range, dynamics, phrasing, and perceived quality unconstrained. This is the software analogue of design by contract [23]: it records the part of the relationship that can be checked without pretending to specify the entire artifact.

2.2 The Counterexample

Executing four cycles yields a multiset of exact rational events. For the first kick, the whole event interval begins at 1/8 rather than 0. The checker returns a proof object of the following conceptual form:

Table 1. The running counterexample is a typed proof object, not a prose error.

Field	Value and role
Verdict	VIOLATED; a concrete witness exists.
Obligation	`kick.grid`: begin times must be 0 mod 1/4.
Witness	Track `kick`, event e_0, whole begin 1/8.
Expected	Nearest legal grid points 0 and 1/4.
Source	Span containing `late(1/8)` and its AST node identifier.
Cause	Derivation path from Mini token `bd*4` through the timing transform.
Preserve	Tempo, bass range, and bass count currently satisfy the contract.

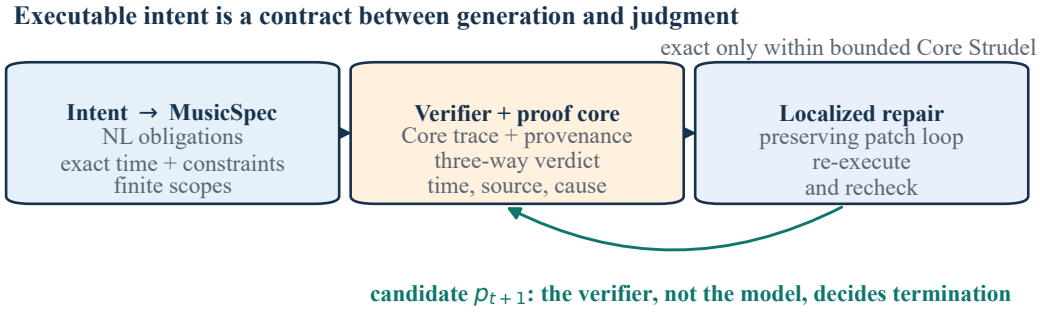

Fig. 1. StruSpec turns checkable intent into an execution contract. A model proposes candidates, while deterministic execution and verification decide whether the loop terminates. Unsupported constructs and incomplete evidence produce UNKNOWN, not success.

A patch model now faces a different problem. It need not infer which promise failed or search the full program for a plausible cause. It can remove or change the localized transform. Crucially, deleting the bass track would not be accepted as a shortcut: the preservation set and the full contract are rechecked after execution. The proof object therefore plays the role that a failure-inducing input, slice, and test oracle play in conventional debugging [35, 38], but it is specialized to exact temporal events.

3 StruSpec

Figure 1 shows the workflow. Natural-language interpretation and patch proposal remain fallible learned components. Admission, execution, checking, and termination are deterministic within a documented boundary. This separation lets the system exploit LLM flexibility without assigning an LLM the authority to certify its own result.

3.1 MusicSpec: Partial but Executable Intent

A MusicSpec is a typed collection $S = \{c_1, \ldots, c_m\}$ of hard obligations. Version 0.1 includes constant tempo, onset grids, MIDI pitch ranges, and event counts. These families were chosen because they are common in the collected requests, objectively decidable over finite event traces, and diagnostically distinct. Each c_i has a stable identifier, a track selector, a finite scope, and family-specific parameters. A request that is contradictory or too underspecified is marked `needs_clarification`; the parser does not silently invent a checkable intention.

Temporal units are explicit. Meter, the beat unit used to state tempo, the cycle unit used by the program, and beats per cycle are separate fields. In particular, the implementation never assumes that one cycle is one bar. Scopes are half-open intervals, and all time quantities are reduced rationals. These choices avoid floating-point boundary errors and make a grid witness reproducible.

Table 2 states the checking rules. Let E_q be selected events whose *whole* begin time falls in scope q. Clipping an event's active interval at a query boundary does not create a new onset. Tempo is checked against independent execution-clock evidence rather than against the requested value copied into the specification.

3.2 A Bounded Executable Language

Arbitrary Strudel is JavaScript and is therefore too broad to execute as an untrusted oracle. Core Strudel is a conservative allowlist: one side-effect-free expression; deterministic pattern constructors; Mini notation; fixed control values; named voices; and documented structural and timing

Table 2. MusicSpec v0.1 checks four obligation families over exact traces.

Family	Satisfaction condition	Counterexample content
Tempo	Execution clock equals requested rate and beat unit.	Observed and expected clock evidence; missing clock yields UNKNOWN.
Onset grid	$\forall e \in E_q$: $(\text{begin}(e) - o) \bmod g = 0$.	Off-grid event, exact begin, offset o, step g, neighboring legal points.
Pitch range	Every selected pitched event lies in $[l, u]$.	Event pitch, violated bound, time, track, and derivation.
Event count	Onset count in each requested window obeys $=, \leq$, or $\geq k$.	Window, observed count, expected relation, contributing events or missing-count proof.

transforms. Arbitrary JavaScript, dynamic property access, external input, side effects, unseeded randomness, and unsupported combinators are rejected before runtime.

The boundary is also quantitative. Source length, AST depth, call count, query horizon, event density, and intermediate products are capped. A program outside the boundary is not rewritten to look supported. It receives a typed admission failure or UNKNOWN. This follows the spirit of bounded model checking [9]: the claim is exact for a finite horizon and an explicit fragment, rather than suggestive for an unbounded language.

Accepted programs run through version 1.2.6 of the pinned official `@strudel` runtime. Queries use its direct `Pattern.query` interface. The adapter normalizes official events but does not approximate their timing. Instrumented AST nodes receive stable SHA-256-derived identifiers; Mini tokens retain native spans. This makes the observed event the authority and the source derivation an attached explanation.

3.3 Trace and Provenance Semantics

For program p and finite horizon $H = [h_0, h_1)$, execution produces a multiset

$$T_H(p) = \{e_1, \ldots, e_n\}. \tag{1}$$

Each event is

$$e = (I^{\text{whole}}, I^{\text{active}}, v, \tau, \pi), \tag{2}$$

where the two intervals have rational endpoints, v is the payload, τ is the semantic track, and π is a derivation DAG. The whole interval is the event before query clipping; the active interval is what overlaps H. For onset and count properties, membership is determined by $\text{begin}(I^{\text{whole}})$ in a half-open scope. This prevents an event sustained across the left boundary from being counted as a fresh onset.

Provenance records the constructor node, Mini token when present, and the sequence of combinators that shaped the event. Structural operators union or map derivations; timing operators attach the transformation and exact parameter; replication creates distinguishable derived nodes. Source spans are explanatory metadata, never a substitute for trace semantics. Removing all provenance fields from an instrumented event yields the corresponding official event under the documented normalization.

3.4 Three-Valued Checking and Proof Cores

The verifier is

$$V(S, T_H(p), \kappa) \in \{\text{SAT}, \text{VIOLATED}, \text{UNKNOWN}\}, \tag{3}$$

where κ contains evidence such as the execution clock and coverage completeness. A concrete witness establishes VIOLATED. SAT requires both no witness and complete evidence for the relevant scope. Missing tracks, unsupported payloads, truncated traces, absent clock evidence, or exhausted resource budgets yield UNKNOWN. Conjunction follows strong Kleene logic: one violation makes the contract violated; otherwise any unknown prevents a satisfaction claim.

For each violated obligation, the checker constructs a proof core

$$\phi = (c, W, \Delta, \Pi, P), \tag{4}$$

where c is the obligation, W a sufficient witness set, Δ the exact expected–observed delta, Π the source derivation, and P the obligations currently satisfied. Witness minimization is family-specific. A pitch violation needs one out-of-range event; a count surplus retains the smallest set that proves the count exceeds its bound; an insufficient count identifies the empty or deficient window rather than fabricating an event. The design resembles counterexample-guided synthesis [14, 30], but the proof object is meant for debugging as well as logical refinement.

3.5 Verifier-Terminated Repair

Given failed program p_0, generic feedback g, and proof core ϕ_0, the two experimental methods receive the same candidate and runtime budgets. The localized method serializes the violated obligation, exact witness, expected delta, relevant source derivation, and preservation set. The generic arm gets a matched high-level failure message but none of those typed fields. At iteration t, the model proposes candidate p_{t+1}. The controller then:

(1) parses and admits p_{t+1} to Core Strudel;
(2) executes it over the frozen horizon;
(3) checks the complete contract and every previously satisfied obligation;
(4) accepts only if all hard obligations are SAT;
(5) otherwise emits the next feedback object while budget remains.

The model cannot terminate by saying "fixed," and a patch cannot improve the target while regressing a measured invariant. This is a small but important change in authority. Conventional LLM refinement uses execution feedback [7, 16]; StruSpec makes the oracle, admissible evidence, and stopping condition explicit.

The implementation also includes deterministic repair rules for typed local faults: replacing a constant tempo, removing a simple fixed offset, clamping a literal pitch, or correcting a literal repeat count. A symbolic-first policy tries these rules and falls back to the same LLM budget if no verified patch is found. The symbolic stage is never credited until official re-execution and full-contract checking succeed.

4 Establishing Trust in the Oracle

An impressive repair rate is meaningless if the checker certifies the wrong programs. We therefore treat semantic trust as an empirical argument built from independent failure modes, not as a single coverage percentage. The validation ladder in Table 3 moves from hand interpretation to implementation differential, property-based transformations, generated oracles, and reviewed natural traces.

Table 3. Validation triangulates independent sources of error. All counts are frozen before the confirmatory repair analyses.

Layer	Population	Result and fault class addressed
Blind semantic gate	50 gold cases, two independent reviewers	First pass 45/50 exposed documentation ambiguities; prospective clarification; second blind pass 50/50.
Runtime differential	100,000 generated programs, 34 constructs	99,999 supported traces matched exactly; 925,449/925,449 events matched; one budget UNKNOWN.
Metamorphic testing	1,200 cases, six relation families	1,200/1,200 relations passed across 2,600 independent queries.
Checker oracle	500 family-balanced cases	500/500 verdicts, including 125 temporal-boundary and 108 source/causal checks.
Natural review	96 outputs, 240 obligations	Reviewer/checker verdict agreement on all cases; 33/33 event-bearing localizations contained the reviewed source interval.
Provenance ablation	108 source/causal oracle cases	108/108 localizations with provenance, 0/108 after removing it.

4.1 Independent Semantics and Runtime Differential

The official runtime is the behavioral reference, but comparing the adapter to it alone could preserve a shared interpretation bug. We therefore implemented a second finite semantics that imports no Strudel package. A frozen generator produced 100,000 broad programs spanning all 34 admitted constructs. Of these, 99,999 completed within budget. Official and independent executions agreed on every normalized trace and all 925,449 events; the remaining program exhausted the independent budget and was classified UNKNOWN. No case was silently dropped.

The test generator also supplies six metamorphic families—including structure-preserving rewrites, rational time shifts, and scope decompositions. All 1,200 cases passed, exercising 2,600 independent queries. Metamorphic testing is valuable here for the same reason property-based testing is valuable elsewhere [8]: it checks semantic relations across a large space without requiring a hand-authored expected trace for every program.

4.2 Checker and Provenance Validation

The generated checker oracle contains 500 cases, 125 per constraint family. Within each family, 50 are satisfying, 50 violated, and 25 deliberately unknown. The checker matched all 500 expected verdicts. The suite includes 125 boundary cases for half-open scopes, clipped active intervals, rational grid offsets, and clock evidence. A separate 108-case suite checks source intervals and causal derivations.

Localization can look plausible even when it merely guesses a nearby token. We tested necessity by erasing provenance while leaving traces unchanged. All 108 causal oracle cases localized correctly with provenance and none did without it. On 96 natural model outputs containing 240 annotated obligations, independent review agreed with checker verdicts. For all 33 constraints with a concrete event and reviewed source interval, the reported interval contained or matched the reviewed source. These results do not prove every future localization correct; they establish that current claims depend on explicit derivations rather than formatting heuristics.

4.3 Why the First Blind Gate Was Not Perfect

The initial independent semantic review agreed on 45 of 50 cases. The five disagreements concerned ambiguities in the written rules—for example, whether scope clipping created an onset—rather than post-hoc changes to observed program behavior. We retained the first review, clarified the rules prospectively, and ran a second independent blind review, which achieved 50/50. Reporting both passes matters: the first is evidence that the specification was not self-explanatory; the second is evidence about the clarified contract. Erasing the first would turn a useful design failure into an implausibly clean story.

5 Benchmark and Study Design

5.1 Benchmark Construction

StrudelSpecBench starts with 1,000 controlled semantic blueprints balanced across the four obligation families. Each blueprint has a reference MusicSpec, a reference Core program, a verified single-fault variant, and five meaning- preserving natural-language paraphrases, yielding 5,000 controlled prompts. Reference programs are SAT; injected faults are independently verified to affect the intended family; proof cores are minimized before release. An additional controlled OOD set varies composition structures while preserving the obligation grammar.

Natural coverage comes from two sources. Held-out model-natural programs are generated from prompts that do not appear in the repair training or rule construction process. Expert requests were collected from 46 contributors: 332 unique requests containing 1,660 adjudicated intended requirements. All 1,244 associated rendering tracks executed in the intake pipeline. Two annotators independently transformed the requests into MusicSpecs with an adjudication step; the resulting request is called "expert-natural," not "ground truth user intent," because contributors did not reconfirm every post-adjudication field.

This mix serves different purposes. Controlled cases identify whether a method can repair a known semantic fault. Model-natural cases test failures not created by a mutation rule. Expert requests reveal interface and namespace problems that a synthetic benchmark can hide. We keep these strata visible rather than treating a pooled average as universal.

5.2 Studies and Endpoints

Table 4 summarizes the completed studies. Internal hypothesis labels are included only to map results to frozen evidence artifacts; the paper is organized by research question.

Compilation gap (H3). We sample 100 controlled and 100 expert-natural prompt blueprints, balanced 25 per primary family in each source. Four frozen profiles—DeepSeek V4 Pro, GLM-5.2, GPT-5.6-terra, and Claude Sonnet 4.6—each receive every prompt five times, for 4,000 logical generations. The parser, Core admission policy, runtime, verifier, prompt budget, and output schema are shared. The funnel records strict parse, syntax compilation, complete Core trace, unconditional all-constraint satisfaction, and semantic violation among complete traces.

Localized repair (H1). The primary population contains 600 failing paired units: 152 controlled IID, 153 controlled OOD, 153 held-out model-natural, and 142 expert-request units, balanced at 150 per primary family. Both conditions use the same GLM-5.2 repair model, three candidate opportunities, 2,400 output tokens, and three runtime calls. Method order is balanced. The primary endpoint is strict-ITT verified repair; success requires independent official re-execution and SAT. A protocol-matched replication reuses the exact frozen units, budgets, method order, and endpoint with official DeepSeek V4 Pro.

Table 4. Completed studies. Strict ITT keeps terminal infrastructure and protocol failures in the assigned denominator.

ID	Question	Frozen design	Primary endpoint
H3	Compilation gap	200 prompts × 4 generator profiles × 5 replicates; controlled and expert-natural balanced by family.	Validation funnel; violation among complete traces.
H1	Localized repair	600 mixed-source failing units, paired generic vs. localized; GLM primary and DeepSeek replication.	Strict-ITT verified repair risk difference.
E9	Expert interface recovery	192 new author-confirmed requests; paired generic vs. localized under GLM and DeepSeek; all semantic-entry failures retained.	Strict-ITT satisfaction and conditional repair among complete-trace violations.
H2	Symbolic-first repair	1,200 units; equal-budget LLM patch vs. symbolic then LLM.	Success non-inferiority and normalized AST edit size.
H5	Visible feedback fields	600 controlled faults × token-balanced 2 × 2 witness/location conditions.	Within-unit factorial main effects.
H4	Pre-failure conditioning	200 anchors × 4 profiles × 3 replicates × direct/gold/parsed conditions.	First-attempt SAT against the gold contract.

Prospective expert interface recovery (E9). The original expert 0/142 exposes a pre-semantic namespace failure, so E9 uses a new population rather than rewriting that result. Thirty new contributors author 192 capability-scoped requests, then confirm a gold-free rendering of the executable intent and stable `track_1`–`track_8` role contract. Each initial program is hash-assigned before generation. Every non-SAT item remains in a two-model, paired Generic/Localized schedule with at most eight opportunities; cumulative endpoints are frozen at 1, 3, 5, and 8. Strict-output failure is a paired zero-call ITT failure, not an exclusion.

Mechanism and efficiency (H2/H5). H2 pairs an equal-budget LLM-only patcher with a symbolic-first hybrid on 1,200 independent units: 936 controlled construction faults, 138 held-out model-natural failures, and 126 expert-natural failures. The success component uses a predeclared −5-point non-inferiority margin. Patch size is normalized AST edit distance among paired co-successes; the stronger practical claim requires the hybrid median to be at most 75% of the LLM median. Calls and output tokens are supporting resource measures.

H5 isolates two fields from the full localized protocol. Six hundred controlled construction faults are crossed with minimal witness present/absent and source locations present/absent. Exact user prompts are balanced within unit under `o200k_base`; each condition is single shot. This study does not reproduce the iterative preservation-aware loop, so it tests component sufficiency rather than the whole H1 mechanism.

Pre-failure specification conditioning (H4). Two hundred IID controlled anchors are crossed with four generator profiles, three replicates, and three conditions: direct request, request plus gold MusicSpec, and request plus an automatically parsed MusicSpec. This yields 7,200 generations and

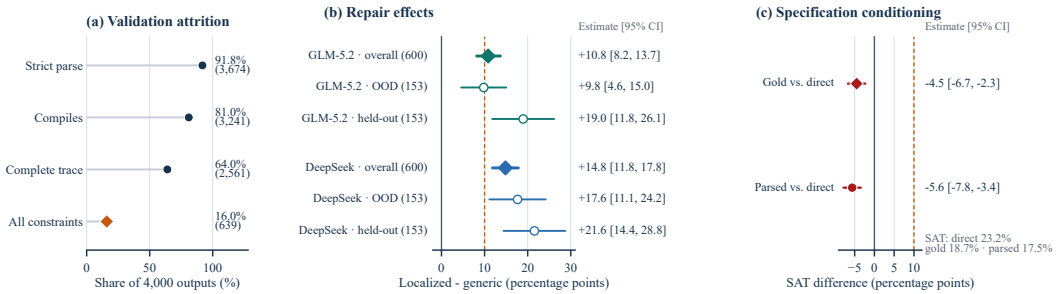

Fig. 2. The semantic gap and the phase asymmetry. Points are estimates; horizontal intervals are blueprint- or anchor-cluster bootstrap 95% CIs. The dashed line in panel (b) is the frozen +10-point practical target; panel (c) shows that even a gold pre-generation contract underperforms direct generation on the controlled population.

600 parser records. Every executable response is re-executed against the unchanged gold MusicSpec. Comparing gold to direct isolates the cost or benefit of a correct redundant representation; comparing parsed to gold measures the additional extraction bottleneck.

5.3 Statistical Analysis

The unit of resampling is the semantic blueprint or anchor, not an individual generation. Percentile 95% intervals use the frozen clustered bootstrap, keeping repeated generations and conditions within their original cluster. H3 uses 5,000 samples; H1 uses 10,000. Paired repair tables are accompanied by descriptive exact McNemar tests. H1 also reports a mixed-effects logistic corroboration with method as a fixed effect and blueprint, generator profile, and primary family as random intercepts.

E9 also uses 10,000 samples but resamples the 30 intact contributor clusters, retaining all requests and both paired methods. It reports GLM and DeepSeek separately. The frozen cross-model criterion requires a positive ITT risk difference in both families; contributor-cluster intervals and exact paired tests remain visible rather than being pooled.

H2 uses paired bootstrap intervals for success and AST distance, a centered bootstrap non-inferiority test, and a sign-flip test for minimality. Its two components form an intersection–union decision; the H2/H5 family follows the frozen Holm procedure. Practical gates remain binding even when a statistical test rejects. H5 estimates within-blueprint factorial main effects. H4 uses anchor-cluster bootstrap contrasts. We report subgroup intervals as descriptive and do not turn them into post-hoc model rankings.

Collection interruptions are handled by whole-prefix exclusion, not outcome selection. H2 excludes all 906 records in a rate-limited high-concurrency prefix before restarting the unchanged schedule at lower concurrency. H4 excludes an entire 172-record pre-amendment prefix after a route change. These records are hash-bound and retained in the provenance ledger. Strict ITT remains primary throughout.

6 Results

Figure 2 puts the three central contrasts on a common scale: validation attrition, post-failure repair effects, and pre-failure specification conditioning.

6.1 RQ1: Compilation Is a Weak Semantic Signal

Of 4,000 logical generations, 3,674 (91.85%) passed strict output parsing, 3,241 (81.03%) compiled, and 2,561 (64.03%) produced complete traces for both required tracks. Only 639 (15.98%) satisfied

every hard obligation. Among complete traces, 1,909/2,561 (74.54%, 95% cluster CI 70.84–78.31) contained at least one semantic violation.

The source strata reveal why compile rate should not be used as a proxy for correctness. Controlled prompts yielded semantic violations in 940/1,591 complete traces (59.08%, 55.00–62.87). Expert-natural prompts yielded 969/970 (99.90%, 99.67–100). The latter number is not a claim that experts write impossible requests. It reflects the greater density of obligations and an interface that required exact named tracks; it is evidence that syntactic and runtime success become less informative as intent grows richer. All four model families meet the frozen semantic-gap rule in both source strata, so the result is not driven by one generator profile.

The practical implication is simple: a compile-only benchmark would count 3,241 apparent successes where the executable-contract benchmark certifies only 639. Compilation remains necessary, but for these programs it is closer to an admission ticket than a correctness judgment.

6.2 RQ2: Counterexamples Improve Verified Repair

With GLM-5.2, Generic-Feedback repaired 31/600 programs (5.17%) and the localized protocol repaired 96/600 (16.00%). The paired difference is +10.83pp (95% cluster CI +8.17 to +13.67), meeting the predeclared +10-point practical target. The paired table contains 24 both-success, 72 localized-only, 7 generic-only, and 497 neither cases (exact two-sided McNemar $p = 1.06 \times 10^{-14}$). A mixed-effects logistic model gives an odds ratio of 6.06 (95% CI 4.60–8.00) for localized feedback.

The protocol-matched DeepSeek replication is stronger in absolute terms: 26/600 (4.33%) generic versus 115/600 (19.17%) localized, a difference of +14.83pp (11.83–17.84). The same-unit model-by-treatment interaction is +4.00pp (0.33–7.67), indicating that magnitude is model dependent while direction remains positive. This is a replication of a treatment contrast, not a model leaderboard.

Generalization is clearest outside the controlled IID construction. For GLM, the localized effect is +9.80pp (4.58–15.03) on controlled OOD failures and +18.95pp (11.76–26.14) on held-out model-natural failures. DeepSeek yields +17.65pp (11.11–24.18) and +21.57pp (14.38–28.76), respectively. Every one of the 577 prior-satisfied obligations in each H1 arm remains satisfied. All 127 GLM and all 141 DeepSeek terminal repair claims independently reverify as SAT.

The expert floor is an interface failure, not a hidden success. The original expert stratum is 0/142 in both arms. Inspection shows that zero units entered with a typed semantic violation: all specifications referenced a track absent from the fixed `kick/bass` output namespace. Thus the comparison stopped at pre-semantic admission. An outcome-blind two-track alignment makes 104 units alignable and 70 both executable and semantically violated. In a prospective follow-up on those 70 units, localized feedback repairs 4 versus 0 with GLM (+5.71pp, bootstrap 1.43–11.43; exact McNemar $p = .125$) and 1 versus 0 with DeepSeek (+1.43pp, 0–4.29; $p = 1$). All five successes reverify, but sparsity precludes a cross-model expert ATE claim. The right conclusion is that alignment removes the absolute zero floor, not that expert-source repair is solved.

A prospective interface recovers the effect across models. E9 closes that mechanism test without altering the old ITT. Its semantic-entry funnel retains all 192 author-confirmed requests: 77 fail the strict output contract, 48 produce incomplete or UNKNOWN Core traces, 67 reach a complete-trace semantic violation, and none is initially SAT. At eight opportunities, GLM repairs 17/192 with Localized and 0/192 with Generic, a +8.85pp difference (contributor-cluster 95% CI 5.50–12.37; exact McNemar $p = 1.53 \times 10^{-5}$). DeepSeek repairs 12/192 versus 1/192, or +5.73pp (3.09–8.51; $p = .000977$). Both prebound directions are positive. Among the 67 complete-trace violations, the paired gains are +25.37pp (16.00–35.71) and +16.42pp (9.38–23.94), respectively. All 30 claimed repairs independently reverify as SAT.

Table 5. E9 paired endpoint at eight opportunities. Brackets are 95% contributor-cluster bootstrap intervals; G and L denote Generic and Localized. Exact McNemar p is 1.53×10^{-5} for GLM and .000977 for DeepSeek in either retained denominator.

Model	Population	G	L	$L - G$ [95% CI]
GLM-5.2	all ITT	0/192	17/192	+8.85 [5.50, 12.37]
GLM-5.2	violations	0/67	17/67	+25.37 [16.00, 35.71]
DeepSeek	all ITT	1/192	12/192	+5.73 [3.09, 8.51]
DeepSeek	violations	1/67	12/67	+16.42 [9.38, 23.94]

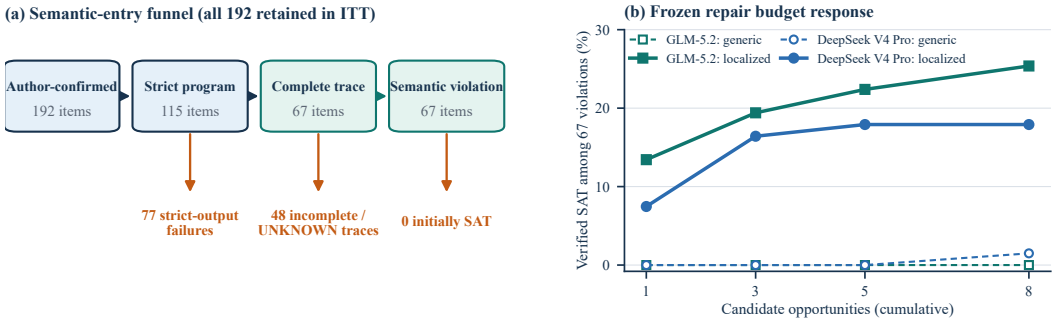

Fig. 3. Prospective expert-interface recovery. (a) The full semantic-entry funnel; orange branches are retained ITT failures, not exclusions. (b) Cumulative verified success on the frozen 67-unit semantic-violation mechanism population. The all-192 ITT effects are reported in text.

Hash-chained exchange replay reproduces all 768 condition terminals, prompts, exact-model routes, and resource totals. Localized also yields fewer candidate preservation regressions than Generic in both models (GLM 43 vs. 69; DeepSeek 22 vs. 38). Because trajectories stop at verified success, this is supporting mechanism evidence, not a separately randomized preservation endpoint. E9 therefore supports a bounded chain—representation, binding, then evidence: author-confirmed roles make localized feedback addressable, while the 125/192 upstream failures show that binding alone does not solve end-to-end generation.

6.3 RQ3: A Narrower Win—Cheaper, Not Typically Smaller

The symbolic-first hybrid repairs 1,094/1,200 units (91.17%), compared with 1,025/1,200 (85.42%) for equal-budget LLM repair. The difference is +5.75pp (4.08–7.50), well above the frozen −5-point non-inferiority margin. It uses 318 model calls and 39,252 output tokens, versus 1,493 calls and 119,687 tokens: reductions of 78.7% and 67.2%. All 1,805 obligations that were satisfied before repair remain satisfied under both methods.

Those gains should not be paraphrased as "smaller patches." Among 1,001 paired co-success units, the LLM-minus-hybrid mean normalized AST distance is +0.00810 (0.00180–0.01546), a small average advantage for the hybrid. But both medians are 0.10, so the hybrid/LLM median ratio is 1.00 rather than the predeclared at-most-0.75 practical target. The strong H2 statement is therefore not supported. A precise summary is more useful: the typed symbolic layer raises pooled success and sharply reduces model use, while leaving the typical successful edit size unchanged.

Population composition matters. All 936 controlled construction faults are solved symbolically with no model call, and the controlled advantage is +6.94pp (5.34–8.65). Held-out model-natural failures show +7.25pp (0.72–13.77). The smaller expert-natural stratum is directionally negative at

Table 6. The completed experiments distinguish semantic value from overclaim.

Finding	Evidence	Licensed claim
Compilation gap	81.03% compile; 15.98% all-hard SAT.	Executability substantially overstates request satisfaction.
Localized repair	+10.83pp GLM; +14.83pp DeepSeek.	The verifier-grounded protocol improves strict repair on the frozen mixed-source population.
Symbolic-first	+5.75pp success; −78.7% calls; equal medians.	Cheaper and more successful in the pooled, controlled-heavy set; not typically smaller.
Field isolation	Witness −0.92pp; location +0.08pp.	Neither displayed field alone explains the iterative protocol effect.
Pre-generation spec	Gold −4.50pp; parsed −5.63pp vs. direct.	A redundant correct contract can be the wrong generation interface.

−4.76pp (−13.49–3.97). The pooled result therefore supports a dispatch policy—use cheap typed rules when their preconditions match, otherwise fall back—not a claim of universal hybrid superiority.

Two visible fields are not the whole mechanism. In H5, the four single-shot repair rates are tightly grouped: 85.00% with neither minimal witness nor source location, 83.67% with location only, 82.67% with witness only, and 84.17% with both. The minimal-witness main effect is −0.92pp (−3.33–1.50); the location effect is +0.08pp (−2.33–2.50). Neither meets its positive-effect gate.

This negative result rules out a tempting but shallow explanation of H1. The repair gain is not an additive reward for displaying a small witness or a line coordinate. H1 changed the entire information and control protocol: a typed obligation, exact delta, execution-grounded causal context, iterative rechecking, and preservation-aware termination. H5 shows that two surface fields are not sufficient on their own; it does not show that provenance is useless, as the source/causal oracle ablation in Section 4 demonstrates.

6.4 RQ4: Correct Structure Can Be the Wrong Interface

Direct generation satisfies the gold contract in 556/2,400 cases (23.17%). Supplying that same gold MusicSpec yields 448/2,400 (18.67%), a paired −4.50pp contrast (−6.71 to −2.29). Supplying an automatically parsed MusicSpec yields 421/2,400 (17.54%), or −5.63pp relative to direct (−7.83 to −3.42). Parsed is only −1.13pp below gold (−2.88 to 0.63).

The parser itself is not good enough: 574/600 outputs (95.67%) are executable MusicSpecs, but only 46/600 (7.67%) are semantically equivalent to the oracle. Executability of a specification is therefore no more a guarantee of faithful intent than compilation of a program. Yet parser error cannot explain the primary generation result. A correct gold specification already reduces success, and parsed and gold conditions are much closer to each other than either is to direct generation.

No generator profile shows the predicted +10-point gold-spec benefit. The largest negative contrast is GPT-5.6-terra at −11.83pp; GLM is essentially flat at +0.17pp. By family, gold structure is descriptively helpful for onset grid (+8.75pp at the constraint level) and pitch range (+4.08pp), but strongly harmful for tempo (−15.58pp). These decompositions suggest that translation difficulty is representation-specific, but they are not confirmatory subgroup claims.

The conservative interpretation is a specification-conditioning cost. These controlled requests already express the relevant constraints. Adding a second, longer schema forces the generator to reconcile two surface forms and translate explicit units, selectors, scopes, and identifiers back into

terse code. That account remains a post-result explanation; identifying it causally would require matched length, field order, and representation ablations. What the experiment does establish is more basic and more consequential: the benefit of a contract after failure cannot be assumed to transfer to the generation prompt before failure.

7 Discussion

7.1 Phase Asymmetry, Not a Universal Structure Prior

The same MusicSpec participates in both H1 and H4, yet the interventions are not the same. Before generation, the schema mostly repeats information that is already present. It may regularize one obligation family and burden another. After failure, execution binds the schema to an observed event, producing a violated obligation, exact delta, and causal derivation that the request alone did not contain. The information state has changed.

This explains why the phrase "structured prompting" is too coarse for the results. The useful object is not structure in isolation but a triangle of *contract, independent execution, and proof*. Remove the contract and there is no explicit oracle. Remove independent execution and the model can grade its own answer. Remove the proof and the repairer receives only a bit. The triangle also explains H5: two fields copied from a proof object do not recreate the control loop that made the proof operational.

7.2 Design Implications for LLM Developer Tools

First, developer tools should distinguish admission, semantics, and quality. A parser or compiler can grant admission; a bounded verifier can decide stated properties; users remain the authority on aesthetic or underspecified goals. Collapsing these layers produces either false confidence or an unusably conservative tool.

Second, repair systems should expose proof objects as stable interfaces rather than compose ad-hoc prose. A typed record can be rendered for an LLM, a human, or a symbolic rule without changing the underlying evidence. It also supports regression checks: a patch can be rejected for breaking an obligation that was previously satisfied.

Third, symbolic rules are best treated as a verified fast path. H2 shows that simple typed faults can be repaired cheaply, but the expert stratum shows the limit of such specialization. A practical dispatcher should test a rule's preconditions, reverify its patch, and fall back without presenting the rule library as general program synthesis.

Fourth, namespace and interface failures deserve first-class status. The expert 0/142 result was caused before semantic checking. A system that reports only final repair success would misdiagnose this as model inability. The validation funnel makes such floors visible. E9 prospectively confirms the remedy: author-confirmed role binding restores a positive localized-repair effect in two model families, while separately exposing representation and execution failures that no feedback treatment can repair.

7.3 Beyond Music

Music is an unusually legible testbed because timing failures can be exact while overall quality remains subjective. The pattern generalizes to other creative or reactive DSLs: animation timelines, shader parameters, robot choreography, data-visualization grammars, and UI motion all mix qualitative goals with objective temporal or structural promises. Transfer requires a domain trace semantics, a bounded contract language, and trustworthy links from observed behavior to source. The numeric effects in this paper should not be transferred without new experiments; the architectural separation can.

8 Related Work

Executable contracts, testing, and semantic oracles. Design by contract makes explicit obligations part of a program interface [23]. Property-based testing samples broad input spaces from declarative properties [8]; symbolic execution and bounded checking systematically explore finite behaviors [5, 9]. All still depend on a meaningful oracle, whose construction is a central testing problem [3]. Recent systems infer contracts with automated validation [32], learn executable intermediate assertions [17], or use temporal properties to drive proof-based repair [31]. StruSpec instead studies the end-to-end authority boundary for generated temporal DSL programs: supported requests and programs are admitted, bounded user obligations are checked, and independent execution witnesses are carried back to source. The oracle is evaluated as a subsystem rather than trusted merely because it is executable.

Program repair and counterexample guidance. Generate-and-validate repair, semantic repair, and learned repair search for patches that satisfy tests or inferred specifications [2, 22, 25, 34]. Surveys emphasize patch correctness, search space, and practical integration [18, 24]. Counterexample-guided inductive synthesis alternates construction with verification [14, 30], while slicing and delta debugging localize causes [35, 38]. StruSpec combines these ideas for a temporal DSL: the counterexample identifies a musical obligation and event, provenance maps it through compositional operators, and the repair loop preserves obligations that were already satisfied. Recent LLM repair work extracts code intent [28], studies which facts should enter the repair context [26], and aligns training objectives with repair rather than generic generation [36]. Fine-grained execution alignment has also been used to localize translation errors [37]. Our causal contrast is different: byte-identical failures hold repair opportunity fixed while the treatment adds a typed obligation, minimal witness, and source-linked proof.

LLM code generation and feedback. Functional benchmarks such as HumanEval and APPS measure execution rather than surface similarity [6, 11]. CodeRL, self-debugging, and self-refinement incorporate execution or critique [7, 16, 20]; external feedback is important because unaided self-correction remains unreliable [13]. Repository-scale benchmarks reveal how much harder repair becomes when context and interfaces grow [15]. Security tests have recently been studied as executable specifications both before generation and during failed-execution repair [19]. That study finds phase-, model-, task-, and coverage-dependent effects. In our temporal DSL, a gold contract can hurt first-attempt generation even though independent execution makes it useful for verified repair. Our proof names the failed obligation, a minimal witness, and its source derivation; a factorial study tests whether those fields explain the protocol. H4 therefore separates executable validity from prompting fidelity rather than claiming that specifications generally harm generation.

Controllable music and live coding. Constraint programming has long represented compositional rules explicitly [1]; neural systems such as DeepBach and Music Transformer target stylistic control and long-range form [10, 12]. Music-generation research also recognizes that structural control and perceived quality are different evaluation problems [4]. Strudel and Tidal provide a compositional live-coding substrate [21, 27]. We do not generate audio directly or claim an aesthetic metric. We verify and repair explicit promises in generated programs.

9 Threats to Validity

Construct validity. SAT means satisfaction of four explicit obligation families over a frozen finite horizon; it does not mean musical quality, user satisfaction, or full intent realization. The system guards against the opposite confusion as well: UNKNOWN is not a violation. H6 listener outcomes are not analyzed in this paper, so no claim is made that formally successful repair preserves perceived

quality. Normalized AST distance is a syntactic minimality proxy, not a measure of cognitive patch simplicity.

Oracle validity. The verifier and the benchmark could share bugs. We mitigate this with a pinned official runtime, an implementation-independent semantics, metamorphic relations, generated reference oracles, natural review, and provenance ablation. The boundary remains only as strong as the documented Core subset, scope conventions, and execution clock. New Strudel versions or constructs require revalidation rather than an assumption of compatibility.

Internal validity. Provider failures, routing changes, and collector concurrency can bias online experiments. We use frozen schedules, exact returned model identities, balanced order, strict ITT, and whole-prefix exclusions made before effect release. H1 includes 131 units with an infrastructure failure in either arm; excluding complete failed pairs raises the GLM estimate from +10.83 to +13.86pp, so the primary result is not created by that sensitivity choice. H5 prompt length is balanced locally; the interpretation of extra provider-reported tokens as invisible routing/security tokens is owner-attested and cannot be independently reconstructed from the provider usage field.

External validity. Four generator profiles and two repair-model families are broader than a single-model study but not the universe of code models. The primary repair population is controlled-heavy. OOD and held-out model-natural effects are positive. The original expert stratum remains blocked by namespace mismatch; the new E9 population restores a cross-model localized effect but is prospectively capability-scoped, and only 67/192 requests reach the semantic repair endpoint. The H4 negative result applies to controlled requests whose constraints are already explicit; it does not show that specifications harm ambiguous natural requests, interactive workflows, or longer planning tasks.

Statistical conclusion validity. Repeated generations are dependent, so intervals cluster by blueprint or anchor. Practical gates are reported alongside statistical tests, including the failed H2 median-size gate. Subgroup results are labeled descriptive. The many exact percentages in this paper arise from frozen finite populations; they should not be read as precision beyond those populations.

Reproducibility and data ethics. Evidence summaries, frozen hashes, analyzers, specifications, and controlled benchmark material are released without contributor identity. Expert requests are anonymized, and claims are limited to adjudicated requirements rather than unverified personal intent. Provider prompts and responses are governed by their collection terms; the public package omits secrets and private endpoint metadata.

10 Conclusion

Compilation answers whether a generated music program can run. It does not answer whether the program kept its promises. StruSpec makes a bounded subset of those promises executable and turns failures into typed, source-linked counterexamples that a repair model cannot overrule. Across two repair-model families, that protocol improves verified repair; a symbolic fast path makes many typed faults much cheaper. A new author-confirmed expert population shows that explicit role binding restores this advantage across both repair models, but its full funnel also makes upstream representation failure visible. Yet the same contract is counterproductive when redundantly placed before first-attempt generation. The deeper lesson is not that structure is always helpful. It is that structure becomes useful when independent execution converts it into evidence.

Data Availability

The repository contains the MusicSpec schema and checker, Core Strudel admission rules, the pinned runtime lock, independent semantics, validation generators, frozen analysis summaries, statistical scripts, controlled benchmark blueprints, and commands used to regenerate the paper figures. Every confirmatory result is linked to a frozen manifest and SHA-256 provenance record. Public expert data are anonymized and contain no contributor identity; provider credentials, private endpoint metadata, and restricted raw journals are not released. The paper can be rebuilt from the tracked source and the official `acmart` 2.19 class included with the artifact.

References

[1] Torsten Anders and Eduardo R. Miranda. 2011. Constraint Programming Systems for Modeling Music Theories and Composition. *Comput. Surveys* 43, 4 (2011), 1–38. doi:10.1145/1978802.1978809

[2] Johannes Bader, Andrew Scott, Michael Pradel, and Satish Chandra. 2019. Getafix: Learning to Fix Bugs Automatically. *Proceedings of the ACM on Programming Languages* 3, OOPSLA, Article 159 (2019), 27 pages. doi:10.1145/3360585

[3] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* 41, 5 (2015), 507–525. doi:10.1109/TSE.2014.2372785

[4] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. 2020. *Deep Learning Techniques for Music Generation.* Springer. doi:10.1007/978-3-319-70163-9

[5] Cristian Cadar, Daniel Dunbar, and Dawson Engler. 2008. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation.* USENIX Association, 209–224. https://www.usenix.org/conference/osdi-08/klee-unassisted-and-automatic-generation-high-coverage-tests-complex-systems

[6] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374

[7] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching Large Language Models to Self-Debug. In *International Conference on Learning Representations.* https://openreview.net/forum?id=KuPixIqPiq

[8] Koen Claessen and John Hughes. 2000. QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming.* ACM, 268–279. doi:10.1145/351240.351266

[9] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. 2001. Bounded Model Checking Using Satisfiability Solving. *Formal Methods in System Design* 19, 1 (2001), 7–34. doi:10.1023/A:1011276507260

[10] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. 2017. DeepBach: A Steerable Model for Bach Chorales Generation. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70).* PMLR, 1362–1371.

[11] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. 2021. Measuring Coding Challenge Competence with APPS. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1.* https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/c24cd76e1ce41366a4bbe8a49b02a028-Abstract-round2.html

[12] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. 2019. Music Transformer: Generating Music with Long-Term Structure. In *International Conference on Learning Representations.* https://openreview.net/forum?id=rJe4ShAcF7

[13] Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large Language Models Cannot Self-Correct Reasoning Yet. In *International Conference on Learning Representations.* https://openreview.net/forum?id=IkmD3fKBPQ

[14] Susmit Jha and Sanjit A. Seshia. 2017. A Theory of Formal Synthesis via Inductive Learning. *Acta Informatica* 54, 7 (2017), 693–726. doi:10.1007/s00236-017-0294-5

[15] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. In *International Conference on Learning Representations.* https://openreview.net/forum?id=VTF8yNQM66

[16] Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. 2022. CodeRL: Mastering Code Generation through Pretrained Models and Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems 35.* 21314–21328. doi:10.52202/068431-1549

[17] Minh Le-Anh, Cuong Chi Le, and Tien N. Nguyen. 2026. Teaching Code LLMs to Reason with Intermediate Formal Specifications. arXiv:2607.04232 [cs.SE] doi:10.48550/arXiv.2607.04232

[18] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. 2019. Automated Program Repair. *Commun. ACM* 62, 12 (2019), 56–65. doi:10.1145/3318162

[19] Yunhao Liang, Chengguang Gan, Ruixuan Ying, Hanjun Wei, Zhe Cui, and Shiwen Ni. 2026. Security Tests as Executable Specifications for LLM Code Generation: Benefits, Trade-Offs, and Coverage Limits. arXiv:2608.09740 [cs.SE] doi:10.48550/arXiv.2608.09740

[20] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-Refine: Iterative Refinement with Self-Feedback. In *Advances in Neural Information Processing Systems 36*. 46534–46594. doi:10.52202/075280-2019

[21] Alex McLean and Geraint Wiggins. 2010. Tidal – Pattern Language for Live Coding of Music. In *Proceedings of the 7th Sound and Music Computing Conference*. 331–334. doi:10.5281/zenodo.849841

[22] Sergey Mechtaev, Jooyong Yi, and Abhik Roychoudhury. 2016. Angelix: Scalable Multiline Program Patch Synthesis via Symbolic Analysis. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 691–701. doi:10.1145/2884781.2884807

[23] Bertrand Meyer. 1992. Applying "Design by Contract". *Computer* 25, 10 (1992), 40–51. doi:10.1109/2.161279

[24] Martin Monperrus. 2018. Automatic Software Repair: A Bibliography. *Comput. Surveys* 51, 1 (2018), 1–24. doi:10.1145/3105906

[25] Hoang Duong Thien Nguyen, Dawei Qi, Abhik Roychoudhury, and Satish Chandra. 2013. SemFix: Program Repair via Semantic Analysis. In *Proceedings of the 35th International Conference on Software Engineering*. IEEE, 772–781. doi:10.1109/ICSE.2013.6606623

[26] Nikhil Parasaram, Huijie Yan, Boyu Yang, Zineb Flahy, Abriele Qudsi, Damian Ziaber, Earl T. Barr, and Sergey Mechtaev. 2025. The Fact Selection Problem in LLM-Based Program Repair. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering*. IEEE, 2574–2586. doi:10.1109/ICSE55347.2025.00162

[27] Felix Roos and Alex McLean. 2023. Strudel: Live Coding Patterns on the Web. In *Proceedings of the 7th International Conference on Live Coding*. Zenodo. doi:10.5281/zenodo.7842142

[28] Haifeng Ruan, Yuntong Zhang, and Abhik Roychoudhury. 2025. SpecRover: Code Intent Extraction via LLMs. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering*. IEEE, 963–974. doi:10.1109/ICSE55347.2025.00080

[29] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 9895–9901. doi:10.18653/v1/2021.emnlp-main.779

[30] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. 2006. Combinatorial Sketching for Finite Programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 404–415. doi:10.1145/1168857.1168907

[31] Yahui Song, Xiang Gao, Wenhua Li, Wei-Ngan Chin, and Abhik Roychoudhury. 2024. ProveNFix: Temporal Property-Guided Program Repair. *Proceedings of the ACM on Software Engineering* 1, FSE, Article 11 (2024), 22 pages. doi:10.1145/3643737

[32] Shubham Ugare, Tarun Suresh, Sasa Misailovic, and Julien Vanegue. 2026. Fun2spec: Code Contract Synthesis at Scale. In *Proceedings of the 34th ACM International Conference on the Foundations of Software Engineering, Companion Proceedings*. ACM, 914–925. doi:10.1145/3803437.3805262

[33] Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, and Yoon Kim. 2023. Grammar Prompting for Domain-Specific Language Generation with Large Language Models. In *Advances in Neural Information Processing Systems 36*. 65030–65055. doi:10.52202/075280-2837

[34] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. 2009. Automatically Finding Patches Using Genetic Programming. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE, 364–374. doi:10.1109/ICSE.2009.5070536

[35] Mark Weiser. 1981. Program Slicing. In *Proceedings of the 5th International Conference on Software Engineering*. IEEE Press, 439–449.

[36] Junjielong Xu, Ying Fu, Shin Hwei Tan, and Pinjia He. 2025. Aligning the Objective of LLM-Based Program Repair. In *Proceedings of the 47th IEEE/ACM International Conference on Software Engineering*. IEEE, 2548–2560. doi:10.1109/ICSE55347.2025.00169

[37] Zhiqiang Yuan, Weitong Chen, Hanlin Wang, Xin Peng, Zhenpeng Chen, and Yiling Lou. 2026. TransAgent: Enhancing LLM-Based Code Translation via Fine-Grained Execution Alignment. *Proceedings of the ACM on Software Engineering* 3, FSE, Article FSE071 (2026), 24 pages. doi:10.1145/3797099

[38] Andreas Zeller and Ralf Hildebrandt. 2002. Simplifying and Isolating Failure-Inducing Input. *IEEE Transactions on Software Engineering* 28, 2 (2002), 183–200. doi:10.1109/32.988498