Compression Without Verification: Self-Distillation and the Reshaping of Continuation Geometry

Flyxion
Independent Researcher

August 2026

Abstract

A recent empirical paper shows that fine-tuning a coding model on its own unverified, temperature-sampled outputs, a procedure the authors call simple self-distillation (SSD), produces substantial and consistent gains on competitive-programming benchmarks [1]. In a separate stress test, the method continues to improve performance even when training-time truncation is removed and most sampled outputs become unusable as complete programs. The authors explain this through a distinction between "lock" positions, where the correct continuation is nearly fixed, and "fork" positions, where several viable continuations compete, and argue that training under one decoding policy and evaluating under another lets the model sharpen locks while preserving forks. This essay accepts the paper's empirical core and its local mechanism, and argues that the mechanism is best understood not as knowledge creation but as a change in continuation geometry: a redistribution of which trajectories a fixed control policy can reach without destabilizing already-established distinctions. Read this way, the paper becomes an unusually clean, real-model instance of several claims developed elsewhere in this research program, including compression after expansion, aspect relegation, admissibility across substrates, and the separation of optimization from warranted belief. The essay also identifies where the paper's own controls stop short of settling the questions those claims raise, and proposes experiments that would close the gap between a plausible mechanism and a demonstrated one.

1 Introduction

A model can improve at a task without being supplied any newly verified answer to that task. This sentence sounds like a paradox only if capability is treated as synonymous with certified information content. Once capability is treated instead as a property of a policy acting on an already-present representation, the result becomes an ordinary description of what can happen when a system is reorganized without being supplied newly verified task information. The paper under discussion supplies a rare, well-controlled, real-model case of exactly this kind of reorganization, and it is worth treating carefully both because the empirical result is strong and because the interpretive move the authors make, from "the model got better" to "the model learned something," is one that this research program has repeatedly found to be too fast.

The procedure is simple to state. A coding model is given roughly ten thousand competitive-programming problems. For each, exactly one answer is sampled at a fixed temperature and truncation setting. No answer is executed, tested, filtered, or compared against a reference solution. The model is then fine-tuned by ordinary supervised learning on this unverified, self-generated corpus [1]. The resulting model is evaluated on LiveCodeBench [2] under model-specific decoding

settings. The temporal construction of LiveCodeBench provides the principal defense against direct pretraining contamination by the evaluation problems, while neither that construction nor internal exact-match deduplication of the prompt pool establishes semantic independence between the SSD training prompts and the evaluation problems, a point returned to in the Critical Assessment section below.

Across six models spanning three families and a range of sizes, this procedure produces consistent and often large gains in pass@1 and pass@5 accuracy on LiveCodeBench, including on its harder problem subsets. The authors' account of why this works rests on a distinction between two kinds of token positions and a claim that training changes the shape of the model's output distribution at each kind differently. The remainder of this essay takes that account seriously, tests how far it can be pushed, and asks what it implies once translated into the vocabulary of continuation geometry, admissibility, and repair that organizes the rest of this program.

2 The Empirical Result, Stated at Its Actual Strength

Simple self-distillation improves every one of the six tested models on LiveCodeBench, with the largest reported gain, for Qwen3-30B-Instruct, moving pass@1 from roughly 42.4% to roughly 55.3%, and the hard-subset pass@1 score rising from 18.3% to 33.6%, an 83.6% relative increase [1]. Hard-subset pass@5 rises from 31.1% to 54.1%, a 74% relative increase. Gains persist at pass@5 and appear on more than one temporal split of the benchmark, which weakens the concern that the result is an artifact of a single evaluation slice. Gains are smaller for models already trained with reasoning-oriented objectives, and some out-of-domain capabilities degrade in the smaller models tested, most visibly a substantial drop on a held-out coding benchmark and a partial loss of correct output formatting on a math benchmark for the smallest model in the study.

Two further results narrow the space of competing explanations. First, sweeping the base model's inference temperature does not reproduce SSD's gains; the base model's pass@1 stays within a narrow band across the temperature range tested, well below the trained model's score. This rules out the simplest deflationary explanation, that SSD merely stumbled onto a better decoding setting that could have been found without any training at all. Second, in a stress-test condition where training-time truncation is removed entirely and the sampling temperature is raised until a majority of generated training samples contain no extractable code and drift into incoherent multilingual output, evaluation-time truncation still recovers a viable output distribution, and the resulting model still shows a substantial gain over the base model, though a smaller one than the standard truncated recipe. Whole-program correctness across the training corpus is therefore not necessary for the corpus to produce a useful update, although the experiment does not establish that each incorrect example contributes positively; it establishes only that a corpus containing mostly non-extractable or corrupted outputs can still be useful in aggregate.

These two results, taken together, are the paper's strongest evidence. They establish that something changes inside the model, not just at its decoding boundary, and that the something is not straightforwardly explained by the correctness of what was trained on. What they do not yet establish is which of several candidate accounts of that internal change is the right one, a point taken up in the Critical Assessment section below.

3 The Proposed Mechanism: Locks, Forks, and Support Compression

The authors' explanation introduces a distinction between two kinds of positions in a generated token sequence. At a lock, the preceding context makes one continuation overwhelmingly likely, such as reuse of an already-established identifier, a contextually forced operator, or the closing symbol of a nested structure, yet the base model still assigns nonzero probability to a long tail of implausible alternatives. At a fork, several continuations are genuinely viable, such as the choice of algorithmic strategy at the start of a solution, and collapsing this genuine multiplicity would be a loss rather than a correction.

Standard inference applies a single temperature to every position regardless of which kind it is. A low temperature protects locks by suppressing their error tail but also collapses exploration at forks. A high temperature preserves fork diversity but reopens the error tail at locks. The authors call this the precision-exploration conflict, and argue that SSD resolves it not by finding a better single temperature but by changing the shape of the distribution the temperature is applied to.

Training on temperature-sampled, truncated self-generations does two things to that distribution. Truncation during data generation removes much of the low-probability tail before it ever enters the training data, so the model is never asked to reproduce it. Temperature during data generation redistributes probability mass among whichever alternatives survive truncation. Fine-tuning on the result pulls the model's own distribution toward this reshaped target: sharply peaked at positions where few alternatives survived, broader and flatter at positions where several did. Ordinary inference, applied afterward at a single fixed temperature, then explores the broad regions without disturbing the sharp ones, because the sharp regions are now sharp in the model's own weights rather than only in the sampling procedure used to generate training data.

The authors summarize this as two coupled operations: support compression, the narrowing of which continuations count as live possibilities, and within-support reshaping, the redistribution of probability among the possibilities that remain, a pairing that recalls earlier work on truncation sampling and neural text degeneration [8, 9]. In the paper's theoretical account, within-support reshaping appears as a Rényi-entropy term whose order is determined by the training temperature [13, 1]. Neither operation by itself is unusual; distillation and temperature sampling are old techniques. What is unusual is the specific coupling, applied through a single fine-tuning pass on unverified self-generated data, and the finding that it substantially improves the reachable accuracy of a fixed decoding policy.

SSD belongs to a longer line of work on knowledge distillation, sequence-level distillation, self-training, and the reuse of model-generated supervision [4, 5, 6, 7]. It is also adjacent to methods that bootstrap reasoning or instruction-following from a model's own generations [11, 12]. Its distinctive feature is not self-generation alone, but the absence of correctness filtering, external feedback, privileged teacher information, or execution-based verification.

4 Two Senses of Compression

The term "compression" operates at two levels in this discussion, and they should not be conflated. The first is support compression in the authors' technical sense: probability mass is concentrated onto the set of tokens retained by the training-time decoding policy. This is a local transformation of a conditional output distribution, produced when a finite set of sampled trajectories is folded back into the weights of the model through supervised fine-tuning. The training corpus is not supplied

to the resulting model as a separate retrieval archive; instead, fine-tuning incorporates structure from the corpus into the parameters, potentially including some sequence-level memorization, while producing a policy that can act on new prompts.

The second is structural compression: distinctions exposed during sampling are reorganized into a continuation geometry in which less exploration is spent on diffuse tails while multiple retained alternatives remain reachable at exploratory positions, across many local support-compression updates taken together. Lower-ranked alternatives become less accessible at precision-bound positions, while several alternatives from the retained head remain accessible at exploratory positions; their retention reflects the model's prior ranking and the selected decoding policy, not an independent judgment that they are correct or intrinsically valuable. This is the sense most relevant to the argument of this essay.

The two levels are related but not identical. Support compression describes the immediate, local training target; structural compression describes the resulting, global change in the organization of reachable trajectories. The title Compression Without Verification refers primarily to the second sense: locally structured information, incorporated into the parameters one support-compression update at a time, can reorganize a model's future behavior even when the complete records carrying that information were never certified as correct.

5 Verification Is Deferred, Not Eliminated

The phrase "without verification" requires a scale index. Individual training trajectories are not verified before admission, and the fine-tuning objective contains no correctness signal. The procedure is therefore verification-free at the level of training-example selection and parameter update. The empirical claim that SSD improves coding performance, however, depends entirely on later execution against hidden test cases. Verification reappears at the level of evaluation.

Three objects must consequently be separated. A training record may be unverified as a complete solution. A local transition within that record may nevertheless preserve syntactic or semantic regularities. The resulting model update may later be validated in aggregate by performance on independently tested problems. Evidence at the third level does not retroactively certify the first: improved benchmark performance does not show that the admitted programs were correct, or even that every admitted program contributed positively.

SSD therefore does not eliminate verification from the epistemic chain. It changes its location and granularity. Correctness is absent when records are admitted and committed, then reintroduced after training as a population-level judgment over the resulting policy. Compression without verification is thus more precisely compression without record-level antecedent verification, followed by aggregate consequential verification.

6 What Kind of Claim Is This?

Before extending the mechanism, it is worth being precise about what has and has not been shown, because the paper's title and framing invite a stronger reading than its evidence supports.

The stronger reading would be that the model has taught itself new problem-solving knowledge without any external signal, a claim that would put pressure on standard accounts of where capability in a trained model comes from. The evidence in this paper does not support that reading.

The gains appear on benchmarks drawn from the same broad domain, competitive programming, that supplied the training prompts. Smaller models lose out-of-domain capability in the same experiments that show in-domain gains, which is what would be expected of a redistribution of an existing capability budget, not what would be expected of the acquisition of new, domain-general competence. And the theoretical analysis the authors provide, an idealized local training objective under an assumed perfect fit, describes how the observed reshaping is consistent with ordinary cross-entropy training; it does not show that a finite network trained on realistic, incomplete trajectories must converge to the lock and fork structure the authors infer from its behavior.

A more defensible reading, and the one this essay adopts, is that the model already represented several viable continuations at each fork and had already assigned most of its probability correctly at each lock, but that no single global decoding policy could exploit both facts at once. The evidence does not require us to posit an expansion of representational content; fine-tuning on ten thousand problem-conditioned trajectories could in principle create new features, associations, or procedural compressions even where the complete answers are self-generated, and nothing in the reported experiments rules this out directly. What the evidence does support is the narrower, more economical interpretation that SSD primarily changes which already-supported continuations a global decoding policy can reach, and it makes that content reachable under one policy applied uniformly across the sequence, rather than requiring a policy that varies with local context in ways ordinary temperature sampling cannot express.

This distinction is the hinge on which the rest of this essay turns; the Continuation Geometry section below restates it in the vocabulary of continuation geometry, and the section on Connections to the Developing Framework traces its wider consequences.

7 Critical Assessment: What the Controls Do Not Yet Rule Out

The paper's central claim, that SSD changes something in the model rather than merely revealing a better inference setting, is well supported by the temperature-sweep control. Several narrower claims built on top of that result are plausible but not yet isolated.

7.1 Missing comparison conditions

The paper does not prominently report SSD against several comparison conditions that would separate its specific contribution from more generic ones: continued training on the original problem prompts without self-generated completions, ordinary language-model training on unrelated code of similar volume, training on default-temperature self-samples rather than the specific temperature and truncation setting used for the main result, and training on a size-matched external synthetic corpus of solved competitive-programming problems, such as rStar-Coder [3]. Without these, it remains possible that part of the reported gain reflects generic domain adaptation to competitive-programming style and vocabulary, rather than the specific lock and fork reshaping the authors describe.

7.2 Evaluation surface and selection

The primary LiveCodeBench v6 evaluation contains only 131 problems, while a larger secondary split, v5, contains 374 [2]; meanwhile the study varies training temperature, evaluation temperature, truncation threshold, checkpoint selection, and model-specific settings across a wide grid, with some reported results selected as the best across checkpoints. Many configurations are therefore being

weighed against a fairly small primary evaluation surface. This does not undermine the qualitative finding that SSD helps, which is too large and too consistent across models to be an artifact of this kind, but it does mean the precise magnitudes reported for any one configuration should be read as optimistic rather than as a stable estimate of what the method delivers under a fixed, pre-registered configuration.

7.3 Contamination controls

The source prompt pool is deduplicated internally by exact matching of whitespace-normalized problem statements, but the paper does not report a semantic or structural similarity analysis between that pool and the evaluation problems. LiveCodeBench's temporal construction reduces the risk that the underlying pretrained model encountered the exact evaluation problems before their release, but temporal separation does not establish that the SSD prompt source lacks earlier problems with closely related statements or solution structures.

7.4 Diversity claims underdetermined by pass@k

The authors treat improved pass@5 scores as evidence that the model preserves solution diversity under SSD rather than collapsing onto a single strategy. Pass@5 can rise, however, simply because each of five independent samples becomes more likely to be individually correct, with no change in how many genuinely distinct algorithmic strategies the model is willing to attempt. Distinguishing these possibilities would require a direct measure of solution diversity, such as clustering generated programs by algorithmic strategy or measuring branch coverage across samples, rather than inferring diversity from an aggregate accuracy statistic.

7.5 The corrupted-data result is under-decomposed

The finding that training on heavily corrupted, largely non-executable self-generations still improves the model is the paper's most striking result, and it is genuinely informative: it shows that whole-sequence correctness is not necessary for the training signal to help. It does not yet show which part of the corrupted data is doing the work. A sample that fails to produce extractable code may still contain a coherent problem-restatement prefix, correctly formatted boilerplate, and locally plausible token transitions before it degrades. An ablation that separately trains on coherent prefixes only, corrupted suffixes only, syntactically valid but semantically wrong completions, and fully shuffled token sequences would locate where the useful signal lives, rather than leaving it distributed across an unexamined mixture.

7.6 Locks and forks are inferred, not independently labeled

The paper's evidence that real-model token distributions change in the direction the lock and fork theory predicts is genuine and is the most direct empirical support for the mechanism. But the classification of a position as a lock or a fork is inferred from the same distributional sharpness that is later used to confirm the theory: sharply peaked contexts are called locks, broad contexts are called forks, and the finding that locks sharpen further while forks stay broad under SSD is then read as evidence for the distinction. This is not circular in the sense of being empty, since the theory does make a falsifiable prediction about how each kind of position responds to training, but a stronger test would classify positions by an independent, semantically grounded criterion, for example whether alternative continuations at that position lead to different but equally valid

downstream programs, before measuring how SSD treats them; work on forking paths in neural text generation offers one starting point for such a criterion [10].

7.7 Cost

The training recipe is algorithmically simple, requiring only ordinary supervised fine-tuning on self-generated data, but it is not cheap: the main experiments use eight high-end accelerators, sequence lengths up to 65,536 tokens, and up to 2,500 training iterations per configuration. "Embarrassingly simple" is an accurate description of the idea, not of the compute required to realize it at the scales tested.

None of these points undermines the paper's central, well-supported claim. They mark the boundary between what has been demonstrated and what has been proposed as an explanation for what was demonstrated, a boundary this essay treats as important to keep visible rather than to round away in either direction.

8 Continuation Geometry: Translating the Mechanism

The lock and fork vocabulary can be restated without loss in terms already developed elsewhere in this program, and doing so exposes what kind of object SSD is actually acting on.

A lock, in this vocabulary, is a narrow continuation seam: a point in a trajectory where a distinction has already been established earlier in the sequence and where any admissible continuation must preserve it. A fork is a region of broader continuation width: a point where several distinct trajectories remain simultaneously recoverable, and where collapsing that width would discard information the sequence has not yet committed to discarding.

Ordinary decoding treats continuation width as uniform across a sequence, because a single temperature parameter cannot distinguish a lock from a fork without external information about which kind of position it is currently at. This is the precision-exploration conflict restated: it is not a conflict between precision and exploration as global goals, but a mismatch between a one-parameter control policy and a continuation geometry that is not one-parameter.

SSD does not resolve this mismatch by adding a second control parameter. It resolves it by changing the substrate the single parameter acts on, so that the substrate itself carries the local structure the control policy lacks. After training, locks are locks in the model's own weights, not only in a sampling procedure applied on top of an unchanged model. A fixed global temperature can then be applied uniformly and still respect local structure, because the local structure is now encoded where the temperature acts rather than needing to be supplied by the temperature itself.

This is the sense in which SSD is best described as a deformation of continuation geometry rather than an addition to represented knowledge. The reachable set of correct programs under a fixed, simple control policy grows, not because the model's raw representational content grew, but because the geometry the policy has to traverse became more compatible with a uniform policy.

9 Connections to the Developing Framework

The eight connections that follow are not of equal evidential weight, and presenting them as uniform subsections would overstate what a single experiment can establish. They are grouped here into four tiers: connections the experiment directly illuminates, a connection that amounts to a strong

mechanistic redescription, a connection that is plausible but more interpretive, and connections that are boundary comparisons rather than supported consequences. The grouping is part of the argument, not a stylistic afterthought; a reader who takes all eight as equally supported has misread the essay regardless of what each individual paragraph says.

Directly illuminated by the experiment.

9.1 The Productive-Constraint Reversal

Truncation during data generation removes possibilities, and yet the resulting model becomes more capable. This is a clean instance of a pattern this program has flagged repeatedly: a constraint that reduces the density of unproductive alternatives can increase, rather than decrease, the space of outcomes that later exploration can productively reach, because exploration is no longer spending its budget distinguishing productive alternatives from a large surrounding field of unproductive ones. The paper's own ablation, that adding truncation on top of temperature reshaping raises the attainable ceiling relative to temperature reshaping alone, is direct quantitative support for this reversal, not merely an illustration of it.

9.2 Admissibility Across Substrates

The paper is an unusually clean instance of the general claim that capability is not equivalent to representational presence. A base model may assign nonzero, even substantial, probability to several distinct viable algorithms for a given problem, and in that sense may be said to represent all of them, while still failing to reach most of them reliably under a decoding policy that is also required to preserve correctness at every lock along the way. What SSD changes is the admissible region: the set of trajectories a fixed global control policy can traverse without destabilizing an already-established distinction elsewhere in the sequence. Narrowing the tail at locks and widening the usable range at forks is, in this vocabulary, a redescription of what it means to enlarge an admissible region under a fixed control budget.

9.3 The Verification Boundary

The paper's clearest limitation, read as a contribution to this program rather than as a flaw in the paper, is that it demonstrates capability improvement without verification while leaving epistemic reliability at the level of any individual output entirely unaddressed. The model becomes measurably better, in aggregate, at producing programs that pass hidden test suites, despite no stage of training knowing which training programs would have passed those suites. This separates optimization from warranted belief in an unusually sharp way: the training signal that improved the aggregate distribution carried no correctness certification for any single training example, and the resulting model's improved aggregate accuracy still does not certify any single one of its outputs. A downstream user selecting one program from the improved model's output distribution gains from SSD's aggregate improvement but gains nothing from SSD toward knowing whether the particular program selected is one of the improved fraction or one of the remainder.

A strong mechanistic redescription.

9.4 Compression After Expansion

SSD's two stages map directly onto expansion followed by non-neutral compression. Sampling at an altered temperature expands the space of continuations the model actually produces, including, in

the extreme condition, continuations that barely qualify as code at all. Fine-tuning then compresses this expanded material back into the same architecture. The compression is not neutral: it does not return the model to its starting distribution, and it does not compress indiscriminately. It relegates the low-ranked tail that expansion exposed and stabilizes the alternatives from the retained head that expansion also exposed, a redistribution of probability mass rather than a verified sorting of good from bad. The round trip changes the geometry of future access even though it begins and ends with a model of the same architecture and, in the corrupted-data condition, is fed material of markedly lower quality than the model's own unperturbed output.

9.5 Record, Admit, Commit

Each self-generated program is recorded during sampling and admitted into the training corpus without any step in the pipeline recognizing whether it is correct. Gradient descent then commits some structure derived from that admitted, unrecognized material into the model's parameters. Only later, at evaluation, does it become apparent that the commitment was, in aggregate, productive. The paper is a demonstration that these four operations, recording, admitting, committing, and later validating, can be and in this case are performed by four different mechanisms operating at four different times, with no mechanism at admission or commitment time having access to the correctness judgment that only evaluation later supplies.

Plausible but more interpretive.

9.6 Aspect Relegation

The low-probability tail suppressed by SSD is not removed from the parameters in any absolute sense; nothing in the procedure guarantees the tail could not be reactivated by a sufficiently different decoding policy or a sufficiently different prompt distribution. What changes is availability under ordinary continuation. Aspects of the model's prior behavior become harder to reach, and other aspects, the alternatives that repeatedly survived the sample-then-train loop, become easier to reach. SSD demonstrates that training can alter which aspects of an existing representational repertoire are foregrounded without requiring the introduction of an externally supplied solution repertoire, which is the operational content of aspect relegation applied to a concrete, benchmarked system rather than to an abstract one; the parameters themselves do change under fine-tuning, so the relevant contrast is not unchanged content against altered access, but a broadly preserved latent capability set against a substantially altered accessibility pattern. This reading is plausible and consistent with the results, but, unlike the three connections above, no reported experiment isolates aspect relegation from the alternative explanations already discussed in the Critical Assessment section above.

Boundary comparisons, not supported consequences.

9.7 Distinction Holonomy

The self-distillation loop begins at one model distribution, generates trajectories under an altered sampling regime, and returns, via supervised fine-tuning, to the same architecture. This is not an identity operation. The round trip leaves a residual transformation: local accessibility relations differ at the end from what they were at the start, despite the start and end points sharing an architecture and task interface. Ordinary learning already guarantees some form of path dependence, so a changed endpoint after a single training loop is not by itself a demonstration of nontrivial

holonomy in the stricter sense developed elsewhere in this program, which would require comparing different generation-and-training paths with matched endpoints or matched data volume and showing a residual transformation that depends on which path was taken. What the paper supplies is best described as a candidate empirical realization of that structure, quantified on a public benchmark, rather than a completed measurement of it.

9.8 Borrowed Intuition, or Its Absence

SSD resembles borrowed intuition in one respect: the student receives a compressed behavioral repertoire without reconstructing, at training time, the reasons any given trajectory succeeds. It differs from borrowed intuition in provenance, and the difference matters. No stronger model, verifier, or curated reference solution set is involved; the repertoire being compressed is generated by the same model rather than transmitted by an external teacher. Calling this borrowed intuition would stretch the term past the distinction it was introduced to mark, between compression earned through the system's own verified experience and compression received ready-made from elsewhere. SSD is better classified as recursive or self-relegated compression: a second compression pass, performed by a model on its own prior output, that changes accessibility without any transmission from an external source. This subsection and the one above are included because they mark a boundary the framework should respect, not because the paper supports them as consequences; keeping that boundary sharp is what lets the framework discriminate between cases instead of merely relabeling them.

10 Discussion: Capability Without Certification

The interpretive risk in a paper like this is not that its authors overstate their numbers, which are reported carefully and with appropriate caveats about scale-dependent side effects. The risk is in how easily a result of this shape gets summarized once it leaves the paper: as a demonstration that models can improve themselves without external information, a framing that quietly substitutes "without a verifier" for "without any external structure at all." The training prompts are external, and they are more than inert scaffolding: they determine the regions of continuation space the model is asked to revisit and resample. The decision of which temperature and truncation setting to sample at is likewise external, made by researchers rather than discovered by the model. What is absent is specifically a correctness signal attached to individual training examples, not external structure in general.

This is worth stating as a named distinction, because it is the conceptual center of the reading offered here: SSD is verification-free but not information-free. The prompts select which regions of continuation space are revisited; temperature and truncation determine how those regions are sampled and recompressed; only the correctness of the resulting complete trajectories goes unchecked. This also specifies what "compression without verification" should and should not be taken to mean. It does not mean that arbitrary noise can be compressed into competence, and the corrupted-data condition does not test arbitrary noise, since it still samples from a model conditioned on real problem statements. It means that structurally selected trajectories, drawn from a fixed and meaningful prompt distribution, can supply a useful training signal without correctness labels attached to the complete trajectories they produce.

10.1 The Sources of Structure

Although SSD supplies no correctness labels, it does not operate on an informationally empty system. At least four sources of structure precede the self-distillation update. The pretrained model already contains statistical regularities derived from its original training corpus. The competitive-programming prompts select a narrow and meaningful region of that prior repertoire. The model's conditional distribution ranks continuations within that region rather than generating them uniformly. Finally, the researcher-selected temperature and truncation policy determines which portions of that ranked distribution become visible to training.

The absence of verification therefore does not imply the absence of selection. Selection occurs before any program is judged correct: through pretraining, prompt choice, conditional probability, and truncation. Fine-tuning then accumulates the local consequences of those selections. What is missing is not structure but a particular kind of structure, namely an external judgment connecting a complete generated program to its task-level correctness.

This distinction explains why the corrupted-data result does not imply that arbitrary noise can improve a model. The corrupted samples remain conditioned on meaningful problems and are emitted by an already-trained coding model. Even where complete trajectories become unusable, their prefixes and local transitions may retain substantial inherited structure. SSD compresses this structured residue, not undifferentiated noise.

Once the substitution between "unverified" and "information-free" is corrected, the more precise and, this essay argues, more interesting claim is available: an existing representation can contain more usable capability than any single global decoding policy can extract from it, and a training procedure that never checks correctness can still change which decoding policy is required to extract that capability, provided the procedure changes the local shape of the model's own output distribution rather than merely the sampling parameters applied on top of an unchanged distribution. This claim does not require and does not receive support for the stronger idea that verification is dispensable in general. It requires only that verification of the training signal and improvement of the aggregate output distribution are separable, which the corrupted-data experiment demonstrates directly.

The formulation that best captures what the paper supports, without extending past it, is this: capability is not exhausted by the continuations a system represents; it also depends on whether a single admissible control policy can traverse exploratory regions of the continuation space without destabilizing precision-bound regions elsewhere in the same trajectory. Self-distillation of the kind studied here improves capability by changing that traversal geometry, compressing unproductive support at precision-bound positions while preserving branching structure at exploratory ones, and it does this without ever certifying which of its own training examples were worth learning from.

11 Limitations and Proposed Experiments

11.1 Falsification Conditions

The continuation-geometric interpretation should be treated as a mechanistic hypothesis rather than as a vocabulary guaranteed to fit every positive result. Several outcomes would count against it.

If independently identified lock positions did not become more resistant to temperature after SSD, the claim that training stabilizes narrow continuation seams would lose its direct empirical support.

If independently identified forks collapsed onto fewer semantic strategies while pass@5 nevertheless increased, the accuracy improvement would be better explained by greater reliability within a narrowed strategy set than by preserved branching structure. If prompt-matched unrelated code produced the same gain as problem-conditioned self-generations, generic domain adaptation would become a sufficient explanation of the benchmark improvement, and the claim that problem-conditioned continuation structure is specifically responsible would lose support. If fully shuffled samples matched coherent prefixes, the proposed role of locally preserved structure would likewise be undermined.

A further challenge would arise if a context-adaptive decoding policy, applied to the unchanged base model, recovered the full SSD gain. The paper rules out ordinary global temperature tuning, but not every policy capable of distinguishing locally narrow from locally broad distributions. If such a policy matched SSD without changing the weights, the result would still demonstrate a mismatch between global control and local continuation structure, but it would weaken the stronger claim that training is needed to reorganize the model's continuation geometry. This yields a three-way distinction worth keeping separate: a global decoder lacks sufficient local control; an adaptive decoder might read and respond to the existing geometry without changing it; SSD changes the geometry itself so that a global decoder becomes sufficient.

These conditions separate the empirical phenomenon, that SSD improves performance, from the particular explanation advanced here. The former may survive even if the latter fails.

The gap between the mechanism as stated by the authors and the mechanism as translated into continuation-geometry terms in this essay is a gap of vocabulary, not of evidence; the translation adds no empirical content the original paper did not already supply. Closing the remaining gap between a plausible mechanism and a demonstrated one would require the comparison conditions and ablations identified in the Critical Assessment section above. Of these, four are proposed here as the most informative given the effort required relative to what they would settle.

First, a lock and fork classification performed independently of distributional sharpness would test the distinction on grounds that do not risk circularity with the sharpness measure used to detect it. This should be defined at the level of semantically distinct continuation prefixes rather than isolated tokens, since at many candidate forks locally different tokens do not correspond cleanly to different algorithms, while a single algorithmic choice is often distributed across many tokens. A workable protocol: generate alternative continuation prefixes from a shared context, cluster them by algorithmic strategy, verify that multiple clusters lead to correct programs, and trace probability mass back to the earliest point of divergence among the clusters. That tests the hypothesis at the scale where "fork" has semantic meaning rather than at the scale where it is easiest to measure.

Second, a decomposition of the corrupted-data training condition into coherent-prefix-only, corrupted-suffix-only, syntactically valid but semantically wrong, and fully shuffled variants would locate which part of the corrupted signal carries the improvement, separating the paper's most surprising finding from an unexamined mixture.

Third, a direct diversity measure, such as clustering sampled programs by algorithmic strategy before and after SSD, would test whether pass@5 gains reflect preserved branching structure, as the fork-preservation account predicts, or a collapse onto fewer strategies each individually more reliable, which would remain consistent with the paper's accuracy numbers but not with its diversity claim.

Fourth, and most directly relevant to this essay's central claim, a four-way comparison would test

the cross-scale bridge this essay's account depends on: the claim that local, next-token imitation of unverified sequences can improve task-level program correctness because of which local regularities the training data preserves, not because of any property of the complete programs as wholes. The comparison separates correctness, local coherence, prompt-conditioned relevance, and generic code exposure as distinct candidate sources of the improvement: one student trained on correct self-generated solutions, one on incorrect but syntactically valid solutions, one on coherent prefixes truncated immediately before failure, and one on prompt-matched sequences drawn from unrelated problems, with token count and optimization steps matched across all four. The reported corrupted-output experiment confounds these four factors; this design would not, and a result in which the truncated-coherent-prefix condition performs nearly as well as the correct-solutions condition would be direct evidence for the local-regularity account of the bridge, as opposed to either "the corrupted data taught it something anyway" or "the geometry did it" left unexamined at the level of mechanism.

12 Conclusion

The paper under discussion offers rare, well-controlled evidence that a model's aggregate accuracy can be substantially improved by retraining it on its own unverified output, surviving even severe degradation of the retraining data, and that its proposed lock and fork mechanism is well motivated though not yet fully isolated from adjacent explanations. Read through the vocabulary developed elsewhere in this program, the result is most economically interpreted as a reorganization of an already-present representational repertoire: without verification of individual training examples, the model is changed so that a fixed global control policy reaches more of the capability its prior distribution already appeared to support. The experiments do not rule out the formation of new features or procedural compressions during fine-tuning, but they do not require such expansion to explain the reported gains. The results support the hypothesis that a system can acquire a more productive continuation geometry from records whose individual correctness was never established because training can respond to recurrent local structure without evaluating the warranted truth of each complete artifact. That is a narrower claim than knowledge creation, better supported by the paper's own controls, and no less consequential for being narrower.

References

[1] Ruixiang Zhang, Richard He Bai, Huangjie Zheng, Navdeep Jaitly, Ronan Collobert, and Yizhe Zhang. Embarrassingly Simple Self-Distillation Improves Code Generation. arXiv preprint arXiv:2604.01193, 2026. https://arxiv.org/abs/2604.01193

[2] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida I. Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and Contamination Free Evaluation of Large Language Models for Code. arXiv preprint arXiv:2403.07974, 2024. https://arxiv.org/abs/2403.07974

[3] Yifei Liu, Li Lyna Zhang, Yi Zhu, Bingcheng Dong, Xudong Zhou, Ning Shang, Fan Yang, and Mao Yang. rStar-Coder: Scaling Competitive Code Reasoning with a Large-Scale Verified Dataset. arXiv preprint arXiv:2505.21297, 2025. https://arxiv.org/abs/2505.21297

[4] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the Knowledge in a Neural Network. arXiv preprint arXiv:1503.02531, 2015. https://arxiv.org/abs/1503.02531

[5] Yoon Kim and Alexander M. Rush. Sequence-Level Knowledge Distillation. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1317–1327, 2016. https://arxiv.org/abs/1606.07947

[6] Tommaso Furlanello, Zachary Chase Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born-Again Neural Networks. Proceedings of the 35th International Conference on Machine Learning, pages 1607–1616, 2018. https://proceedings.mlr.press/v80/furlanello18a.html

[7] Junxian He, Jiatao Gu, Jiajun Shen, and Marc'Aurelio Ranzato. Revisiting Self-Training for Neural Sequence Generation. Proceedings of the Eighth International Conference on Learning Representations, 2020. https://openreview.net/forum?id=SJgdnAVKDH

[8] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The Curious Case of Neural Text Degeneration. Proceedings of the Eighth International Conference on Learning Representations, 2020. https://arxiv.org/abs/1904.09751

[9] John Hewitt, Christopher D. Manning, and Percy Liang. Truncation Sampling as Language Model Desmoothing. arXiv preprint arXiv:2210.15191, 2022. https://arxiv.org/abs/2210.15191

[10] Eric J. Bigelow, Ari Holtzman, Hidenori Tanaka, and Tomer D. Ullman. Forking Paths in Neural Text Generation. Proceedings of the Thirteenth International Conference on Learning Representations, 2025. https://arxiv.org/abs/2412.07961

[11] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping Reasoning With Reasoning. Advances in Neural Information Processing Systems, volume 35, 2022. https://arxiv.org/abs/2203.14465

[12] Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large Language Models Can Self-Improve. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pages 1051–1068. Association for Computational Linguistics, 2023. https://doi.org/10.18653/v1/2023.emnlp-main.67

[13] Alfréd Rényi. On Measures of Entropy and Information. Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, volume 1, pages 547–561, 1961.