WHAT THE MASK STILL BUYS: MEASURING WHICH DECODING CONSTRAINTS SMALL LANGUAGE MODELS INTERNALIZE

Anonymous authors
Paper under double-blind review

ABSTRACT

Schema-derived constraints let a small language model emit valid MLIR at inference time: a token-level grammar mask enforces syntax and type domains, and a scope validator rejects use-before-definition. We ask what the mask still buys once a model has been trained to satisfy it, layer by layer, and whether learning validity brings correctness with it. We train students on their own masked-and-verified rollouts (symbolic self-distillation, SSD) from 135M to 3B parameters in two families and measure, per constraint layer, the violation rate it still removes after training, paired per prompt on frozen pools. Validity moves into the weights: type domains internalize completely, syntax up to a structural residual of 5 to 11 points plus a vocabulary residual that shrinks when the training pool covers the missing concepts, and scope up to a small residual that a per-token scope signal mostly removes at 360M. Correctness follows only where the model already samples it. Verifier acceptance under free decoding rises by up to 70 points. Under a gold-differential harness, masked correctness falls at 135M, 360M and Gemma-1B; at 1.7B and 3B sampled pass@1 rises by 7 to 12 points, while the set of prompts solved within eight samples does not grow. We read both as concentration rather than acquisition. A functional reward does not change this on the released pools. Self-distillation raises the untrained model's pass@1 and lowers its pass@16. Against the 49 of 143 prompts the untrained 360M model solves within 256 masked samples, nine on-policy cells each solve at most one prompt outside that set, while five cells that see a gold program solve 13 to 22, with no overlap between the groups. Supplying one gold program per rollout group restores correctness at a price in validity that an adapter control and a second seed show is not separable from adapter capacity at the scales we compare. As in reinforcement learning with verifiable rewards, self-distillation selects among programs the model already samples: it teaches validity, which the mask makes abundant, and not correctness, which the model must be shown; here the missing program can be supplied and its price measured. Outside the training distribution the student loses validity on held-out StableHLO above 360M while gaining it on a dialect that reuses the training vocabulary, and for Gemma-1B the mask does not repair the loss, leaving the student behind the model it started from with the full stack restored.

1 INTRODUCTION

Small language models can emit compiler intermediate representations when an inference-time mask keeps them inside the language. Rath (2026) showed that a schema-derived stack gates what a 1.7B-parameter model emits as MLIR: a token mask for syntax (C1) and type domains (C2), plus a scope validator (C3), raise verifier-accepted programs from 34.5 to 67.5% on a frozen pool of 200 prompts. The mask runs at every decoding step. We ask which parts of it a small model absorbs into its weights, which must stay at runtime, and what governs the split, a question posed for action masks in policy gradients (Huang & Ontañón, 2022) and not for a layered grammar.

We answer with a decomposition: each constraint layer has a residual, the violation rate it still removes after training, and a layer is internalized when that residual is at most 2 points with a

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

95% interval containing zero. The layers differ in how a decoder can enforce them. Syntax and type domains compile into a token mask; SSA scope needs a validator over the finished program, because the binding it checks is not context-free in the unbounded language; shape consistency (C4) has no enforcement at all and is only measured. We pre-registered a class hypothesis: that training a student on its own masked-and-verified rollouts closes the gap for the mask-enforced layers and leaves a residual for the validator-enforced and unenforced ones (Appendix D). Only its type threshold held, and what replaces it is sharper than the class it displaces. The two mask-enforced layers separate, type domains absorbed and syntax not; the syntax residual divides into a structural part that scale does not close and a vocabulary part that training-pool coverage governs; and the validator-enforced layer leaves a small residual at or below 1B and none above. Section 4 reports every threshold against its outcome. What survives across scales and families is a separation rather than a universal: verifier acceptance moves into the weights, and correctness follows only where the model already samples it. Supervised fine-tuning on an unseen language teaches syntax before semantics, and no intervention Bannihatti Kumar et al. (2026) try, preference tuning included, closes the gap; we measure that limit per constraint layer and per prompt, and show that the source of the supervision governs the split.

The training instrument is symbolic self-distillation (SSD): the teacher is the student's own weights under the C1+C2 mask plus a verifier, and the student minimizes a per-token legal-mass loss toward its masked self together with a group-relative verifier reward. It combines on-policy distillation (Agarwal et al., 2024) with GRPO (Shao et al., 2024) and is the instrument rather than the contribution; baselines at identical prompts, budgets, and seeds are rejection-sampling fine-tuning (RFT; Yuan et al., 2023), GRPO without the mask, and supervised fine-tuning on template golds. We study open-weight instruction-tuned models from 135M to 3B parameters in two families (Allal et al., 2025; Bakouch et al., 2025; Gemma Team, 2025) on one workstation, on the two frozen MLIR pools of Rath (2026) and on new test sets for held-out dialects and textual LLVM IR; validity is scored by the target's verifier and correctness by a gold-differential harness, and claims are scoped to this regime (Section 8).

Contributions.

1. **A per-layer account of what the mask still buys after training.** After self-distillation type domains internalize completely and their mask can be dropped. After self-distillation syntax keeps a structural residual of 5 to 11 points that scale does not close, plus a vocabulary residual that training-pool coverage governs and that a retrained pool removes on arith+func while leaving it untouched on linalg; off-policy supervision inflates the structural part to 26.5 instead. The scope residual is small, positive at or below 1B and negative above, and a per-token scope loss removes most of what remains at 360M where resampling at 3B does not. Shape, which nothing enforces, does not improve and worsens by 4.8 points on linalg. The pre-registered hypothesis and its outcome are reported verbatim (Section 4).

2. **Off the training distribution the mask buys more than before, and sometimes not enough.** Above 360M the trained student loses free validity on held-out StableHLO, and for Gemma-1B the mask does not repair it: with the full stack restored the student is 48 points behind the model it started from, where at 1.7B and 3B the stack repairs the loss in full. The mask therefore insures against what training removed as well as what it missed. The sign is selective: validity is gained at every scale on a control-flow dialect that reuses the training vocabulary. We report this as a measurement and name three explanations we cannot separate (Section 6).

3. **What on-policy training lacks is the program, not the reward, and supplying it has a price.** That supervision from outside the policy reaches problems reinforcement does not is known (Yue et al., 2025; Zhou, 2026); we hold the algorithm fixed and vary only the signal. Against the 49 of 143 prompts the untrained 360M model solves within 256 masked samples, nine on-policy cells each solve at most one prompt outside that set, while five cells that place a gold program inside the same rollout group solve 13 to 22. A reward that runs the gold program and pays for matching it lands in the on-policy group too, so what the student lacks is not information about correctness in the reward but a correct program in its own samples. The expansion is then priced on the second verifiable axis: free validity falls

by 34.5 points at 360M, and the smaller price at 1.7B does not survive an adapter control and a second seed (Section 7).

2 Related Work

Constrained decoding. Grammar-constrained decoding restricts the next-token distribution to the continuations a formal grammar admits, by incremental parsing (Scholak et al., 2021; Poesia et al., 2022) or by compiling the grammar into token-level automata (Willard & Louf, 2023; Geng et al., 2023; Ugare et al., 2025; Beurer-Kellner et al., 2024; Dong et al., 2025); with engine benchmarks, evaluations against fine-tuning, and decoding-time variants (Geng et al., 2025; Roy et al., 2023; Zhang et al., 2023; Wang et al., 2023). Type-constrained decoding extends the mask to a type system (Mündler et al., 2025), and type-guided synthesis reports that a model which internalizes the type system gains correctness as well as validity (Huang et al., 2025); our C2 layer is a finite domain restriction rather than a type system, and internalizing it brought no correctness. Constrained decoding distorts the sampled distribution away from the grammar-conditioned one (Park et al., 2024) and helps base models where it hurts instruction-tuned ones (Schall & de Melo, 2025). An incomplete constrainer pushes code models into low-probability regions, where a type-checker reward raises unconstrained compliance from 72 to 92% without raising functional correctness (Biagiola et al., 2026). Grammars extracted from MLIR dialect definitions drive dialect-agnostic fuzzing (Vaidya et al., 2025); Rath (2026) layered them into the C1 to C3 stack and showed that token-level masks and post-hoc filters are not interchangeable. What is ours is the per-layer measurement of what training absorbs from that stack.

Distillation, constraint losses, and verifiable rewards. Knowledge distillation transfers a teacher's distribution into a student (Hinton et al., 2015); on-policy variants train on the student's own samples (Agarwal et al., 2024; Gu et al., 2024; Lu & Thinking Machines Lab, 2025), and context distillation uses the model itself, given extra context, as the teacher (Snell et al., 2022; Zhao et al., 2026). Our legal-mass loss is the autoregressive semantic loss (Xu et al., 2018; Ahmed et al., 2023) over the grammar's allowed set, and equals as a training term the projection cost Reddy et al. (2026) measure at decoding time; Kung et al. (2026) train under sampled constraints and correct the resulting ratio, which our objective does not. Distilling rules into weights goes back to Hu et al. (2016), and amortizing constrained generation is the aim of Bhagavatula et al. (2023); Wang et al. (2022); Lu et al. (2025); Hu et al. (2024). Rejection-sampling fine-tuning (Yuan et al., 2023; Zelikman et al., 2022) and group-relative policy optimization (Shao et al., 2024) drive reinforcement learning with verifiable rewards (Guo et al., 2025), which raises pass@1 without raising pass@k at large k (Yue et al., 2025; Wu et al., 2025; Kim et al., 2025; Strozzi, 2026; Nicolicioiu et al., 2026) and can lower it on the prompts whose correct trajectories are rare but reachable (Zhou, 2026), because post-training amplifies what pretraining covers (Zhao et al., 2025; Chen et al., 2026; Matsutani et al., 2026; Huang et al., 2025). Several lines argue the boundary can be pushed out with enough compute, a dense warm-up reward, or interleaved supervision (Liu et al., 2025; Ma et al., 2026), which our budget does not test. Proposed escapes reward unlikely samples, train longer, or add easier data (He et al., 2025; Liu et al., 2025; Prakash & Buvanesh, 2025); on-policy updates forget less than off-policy ones (Shenfeld et al., 2026; Chen et al., 2026), self-distillation is proposed for learning new skills without losing old ones (Shenfeld et al., 2026), and post-training gains generalize inconsistently across domains and can vanish (Hu et al., 2026). Mixing off-policy demonstrations into the groups (Yan et al., 2025; Wu & Liu, 2025; Zhang et al., 2026; Ma et al., 2026; Yan et al., 2026) is the mechanism of our one-gold-per-group ablation. Yue et al. (2025) already report that distillation from a stronger teacher reaches problems reinforcement does not, and those papers price naive injection in entropy and disrupted response patterns (Yan et al., 2025; Zhang et al., 2026). We hold the algorithm fixed and vary only the signal, which puts a reward carrying the target semantics on the same axis as a gold program, and we price the expansion on a second verifiable axis, validity, per scale. In small code models a lint-only penalty can lower lint errors without reliably improving correctness (Skopin & Kotelnikov, 2026), while execution-based verification does raise correctness for low-resource languages where the base model already samples working programs (Boruch-Gruszecki et al., 2026). Our SSD is unrelated to the simple self-distillation of Zhang et al. (2026), which uses no verifier and reports pass@1 gains.

Formal languages and transformers. Transformers process bounded hierarchical structure only to a bounded depth (Yao et al., 2021), and the Chomsky position of a task was proposed as a predictor of out-of-distribution generalization (Delétang et al., 2023). Later work replaced the class with finer predictors: length generalization tracks expressibility in RASP-L rather than Chomsky class (Zhou et al., 2024; Huang et al., 2025), and within the regular languages learnability tracks rank and expected string length (Borenstein et al., 2024), though a benchmark of frontier models still finds stratification by class (Dong et al., 2026). Those results mostly concern learning from scratch, and fine-tuning pretrained models on formal languages is itself recent (Ghosh et al., 2026). They also already pointed away from the hierarchy, so our pre-registered class hypothesis was a prior we expected to lose rather than a strawman; what was open was whether a brief fine-tune on a model's own masked rollouts absorbs a validator-enforced check at all. The data do not bear it out (Section 4).

Probes and patching. Probes need controls and patching needs care in metric and interpretation (Hewitt & Liang, 2019; Belinkov, 2022; Heimersheim & Nanda, 2024; Meng et al., 2022). Code models represent syntax, data flow, program semantics and types (Troshin & Chirkova, 2022; Jin & Rinard, 2024; Lucchetti & Guha, 2025) and can encode information they do not surface (Orgad et al., 2025; Sharma et al., 2026); fine-tuning tends to enhance existing mechanisms rather than build new ones (Prakash et al., 2024; Jain et al., 2024), the reading our unchanged probes support.

Compiler intermediate representations. MLIR is a multi-dialect compiler infrastructure with a verifier (Lattner et al., 2021). Models trained on LLVM IR learn its syntax (Cummins et al., 2023; 2024; Paul et al., 2024), while general-purpose models parse IR but struggle with instruction-level execution reasoning (Jiang et al., 2025). Verifier-rewarded reinforcement learning has produced correct IR rewrites at 3B (Fang et al., 2026) and correct Verilog and Triton kernels (Wang et al., 2025; Woo et al., 2026); Section 7 says why our setting differs. Supervised fine-tuning teaches the syntax of an unseen language before its semantics, and no intervention Bannihatti Kumar et al. (2026) try at 4B to 32B, preference tuning included, closes the gap; verifier-reward training does not close a planning model's cross-domain gap (Belcamino et al., 2026). We measure that limit per layer and per prompt under mask-derived supervision and price the missing program. Correctness is judged by execution (Yang et al., 2011; Lopes et al., 2021; Chen et al., 2021).

3 SETUP

Constraint layers and stacks. The generation grammars and lattices are derived from the compiler's operation definitions. C1 is a token-level mask from a syntax-only grammar; C2 adds type-domain splitting to the same grammar; C3 is a validator applied as rejection sampling with up to five seeded attempts; it rejects use-before-definition and also cross-operation type mismatches, so it is not a pure scope check and its residual bounds both; C4 is shape and interface consistency, which has no mask and is only measured (Appendix B). The four stacks are FREE, C1, C1+C2, and C1+C2+C3.

Models. SmolLM2-135M, 360M, and 1.7B-Instruct and SmolLM3-3B form the primary ladder; Gemma-3-270M and Gemma-3-1B are the family control with a different pretraining mix and a 262k-token vocabulary; both training recipes collapse on Gemma-3-270M (Appendix C), so it appears in the zero-training ladder only. Instruct checkpoints are used throughout with the three-shot prompt of Rath (2026), pinned by revision.

Pools and test sets. Evaluation uses the two frozen pools released by Rath (2026), arith+func (n=200) and linalg (n=125), never used for model selection. Functional correctness is scored on the 143 arith+func and 30 linalg prompts that pass the gold gate of the differential harness. Transfer uses the released StableHLO sets and two new hand-authored sets, Scf-Spec-60 and LLVM-Spec-60 (Section 6).

Training pool. A template generator over the same operation families produces prompts with gold programs (7,257 train and 500 dev after deduplication); no training gold is alpha-equivalent to any test gold, and 8-gram overlap with the 790 protected records is zero. Golds enter on-policy training only through the reward (datasheet in Appendix A).

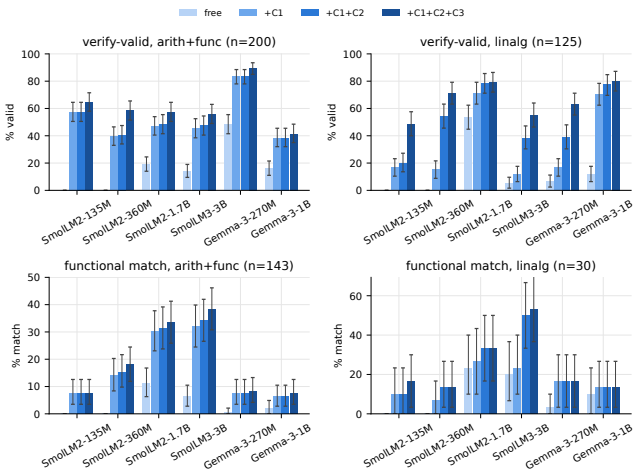


Figure 1: Zero-training constraint ladder for the six models on both frozen pools (verify-valid and functional). Bars are rates over the full pool (n=200 arith+func, n=125 linalg)

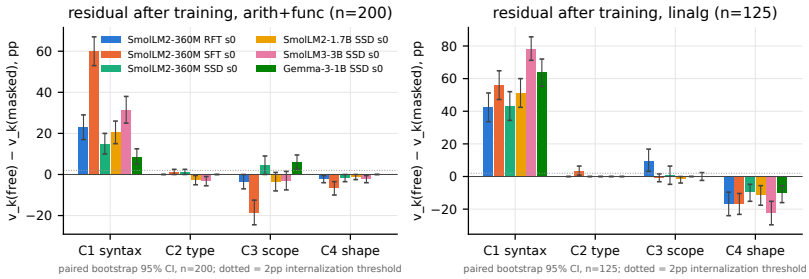


Figure 2: Per-layer residual after training with paired-bootstrap 95% CIs, per checkpoint and pool.

Metrics. For model M and stack S_k, $v_k(M, S)$ is the free-decoding violation rate of layer k: parse-invalid (v_1), type-domain error (v_2), scope failure (v_3), shape or interface failure (v_4). The gap $G_k(M) = \text{rate}(M, S_k) - \text{rate}(M, S_{k-1})$ is what layer k buys at inference before training; the residual $R_k(M') = v_k(M', \text{FREE}) - v_k(M', S_k)$ is what it still buys after training M into M'. A layer is internalized when the paired-bootstrap 95% CI (10,000 resamples) of R_k contains zero and the point estimate is at most 2pp. The type, scope, and shape flags are defined on parse-valid programs, so their residuals can be negative when free decoding rarely parses. Functional correctness is the composed rate of programs matching the gold on five randomized trials, with non-conforming interfaces counted as failures.

Compute. Every experiment runs on one Apple-silicon workstation with the MLX stack and a pinned LLVM 19.1.7 verifier; full fine-tuning up to 360M parameters, rank-32 adapters (Hu et al., 2022) above.

4 The Constraint-Class Decomposition

4.1 Zero-training ladder

Figure 1 and Table 3 (Appendix D) extend the ladder of Rath (2026) down to 135M, up to 3B, and across to the Gemma family. Free decoding at 135M and 360M produces no verifier-accepted program on either pool, so there the mask buys the whole gap. The type layer carries up to 39pp on linalg and almost none on arith+func, where the grammar already fixes most type slots.

4.2 RESIDUALS AFTER TRAINING

Figure 2 and Table 6 (Appendix D) give the per-layer residual for every trained checkpoint. Three regularities hold for the SSD students across 360M, 1.7B, 3B, and Gemma-1B. First, the type-domain residual is zero on linalg at every scale, with a paired interval of $[0, 0]$ against an untrained gap of up to 39pp, and at most 1pp on arith+func: the type layer is internalized. Second, the syntax residual is not zero anywhere, ranging from 8.5 to 31.5pp on arith+func, and it splits into two parts with different causes. The structural part covers cast form, whitespace-broken comparisons and malformed shapes; it stays in a narrow 5 to 11pp band at every scale and does not shrink, while off-policy supervision inflates it to 26.5pp. The vocabulary part is invented mnemonics such as `arith.allocf` for concepts the training pool never showed; it grows with scale under adapters and dominates linalg, but a full fine-tune at 1.7B cuts it to a quarter of the adapter value, so its growth is partly an adapter effect (Section 8). Third, the scope residual as defined in Section 3 is gated on parsing and goes negative at 1.7B and 3B. Conditioned on prompts whose free output parses, an exploratory definition adopted after the pre-registered one failed, the scope residual is positive but small at every scale, from 7.7pp at 360M to 1.5pp at 3B. Its intervals exclude zero only at 360M and Gemma-1B and overlap across scales, so the decrease with scale is a point-estimate trend rather than a resolved one (Table 6 in Appendix D). At 3B the free scope-failure rate is still 27% and five validator retries leave 25.5%, beyond what rejection sampling repairs. The shape layer, which nothing enforces, does not improve at any scale and worsens by 4.8 points on linalg.

Framing gate. The pre-registered thresholds (Table 5 in Appendix D) asked for syntax and type residuals with CIs containing zero and a scope residual positive at every scale and decreasing with scale. Only the type threshold is met, so the class hypothesis fails in its pre-registered form in this stack. What the data support is narrower: self-distillation moves validity on its training distribution into the weights, type domains completely, syntax up to a structural residual and a coverage-bound one, scope partially, and shape not at all. Off-policy supervision is the exception to the type result, at $+3.2$ on linalg.

A per-token scope signal closes most of the residual. A synthetic SSA language with a tunable dependency distance shows no distance-driven residual at these sizes (Appendix D), so the scope residual rejection sampling leaves could be a limit of the student or of the signal. Adding the offline scope oracle as a per-token mass loss to SSD at 360M (Appendix B) answers this: the free scope-failure rate falls from 13.5 to 8.0%, the scope residual from 4.5 to 1.0 points (parse-conditioned 7.7 to 4.0), free-decoding acceptance rises from 69.5 to 77.5%, and correctness does not move (24, 23, 23, and 23 of 143; Table 10). The scope layer resists the sequence-level signal, not the weights.

Coverage governs the syntax residual. To test the vocabulary claim causally we add fourteen template families for the concepts on which the students invent operations (Appendix A) and retrain the 360M student with the same recipe. On arith+func the syntax residual falls by nine points, most of it vocabulary, and free acceptance rises to 82.0%, in all three seeds (Table 9). On linalg, which the new families do not cover, the residual stays at 42 points, and correctness does not move (Table 10 in Appendix D). Coverage buys validity, not semantics.

What the mask still buys is therefore narrower than a class law: the type mask can be dropped, the syntax mask buys a structural residual scale does not close plus a coverage-governed vocabulary one, and the validator buys a few points a per-token signal mostly removes.

5 SYMBOLIC SELF-DISTILLATION

Objective. Let π_θ be the student and A_t the set of tokens the C1+C2 grammar admits after the sampled prefix $x_{<t}$. The masked self π_θ^{mask} is π_θ restricted to A_t and renormalized. The reverse Kullback-Leibler divergence from π_θ to π_θ^{mask} is infinite whenever any mass sits outside A_t, so the trainable per-token form is the legal-mass loss: the negative log of the probability the student puts on the allowed set at each step, in expectation over the student's own masked samples (Appendix C). It is the on-policy-distillation implicit reward with the teacher set to the masked self. A sequence-level reward enters through a group-relative advantage: $\mathcal{L}_{SSD} = \mathcal{L}_{\text{mask}} + \lambda\, \mathcal{L}_{GRPO}(r)$ with $\lambda = 1$, plus a frozen masked-base teacher term. The reward term applies the advantage to tokens sampled from

the masked policy without an importance correction, unlike Kung et al. (2026); the omitted term is the legal-mass gradient the first term already carries. Hyperparameters and the derivation are in Appendix C.

A verifier-only reward has a prompt-independent optimum, an early run converging to one always-valid program, so the reward gates on the verifier and grades the program's signature and operation multiset against the template gold.

Baselines. RFT samples under the same mask, keeps rollouts with positive reward, and applies the supervised loss; GRPO samples without the mask and uses the reward only; SFT-gold trains on template golds off-policy. All methods share prompts, rollout budget (2,000 prompts, two epochs, four samples per prompt), and seeds.

Table 2 and Section 4 compare the methods; what separates SSD from RFT at a matched budget is stability across seeds rather than the point estimate, and GRPO without the mask does not learn at 360M, where no free rollout of the base parses. At 1.7B it learns less than SSD at a matched budget, 32.5 against 57.5% free acceptance with a syntax residual of 59.5 against 20.5 (Table 6): the mask, not the reward, moves most of the validity into the weights.

6 Internalization versus Portability

We train on arith, func, and linalg and test on unseen targets: the released StableHLO Spec-30 and Held-Out-200 sets; a structured-control-flow dialect (scf) whose regions stress the scope layer, with a newly derived grammar, a region-aware validator, and Scf-Spec-60; and textual LLVM IR with its own grammar and validator, scored by `lli` on LLVM-Spec-60 (Appendix B). Table 1 compares the trained student with the untrained model under free decoding and under each target's full stack. Verbatim copying is ruled out (Appendix B).

		HLO-Spec-30		HLO-Held-Out-200		Scf-Spec-60		scf correct		LLVM-Spec-60		LLVM correct	
model		free	full	free	full	free	full	free	full	free	full	free	full
360M	base	0	63	0	87	0	10	0	2	0	60	0	5
360M	SSD	40	70	50	100	20	30	3	5	47	60	5	5
1.7B	base	33	67	42	72	17	30	6	9	15	52	4	17
1.7B	SSD	27	73	14	96	40	55	9	11	20	43	10	16
1.7B	+ KL	27	60	31	67	40	55	9	11	35	47	13	15
3B	base	17	83	98	96	15	33	8	12	25	55	14	23
3B	SSD	30	77	44	96	20	42	9	13	20	35	6	8
G-1B	base	27	50	90	96	7	27	1	6	5	22	2	2
G-1B	SSD	43	50	28	48	63	72	5	5	43	45	3	3

Table 1: Portability to unseen targets. Verify-valid rate in percent under free decoding and under the target's full constraint stack, for the untrained model and the seed-0 SSD student trained on arith, func, and linalg. HLO is StableHLO. Correct counts programs matching the reference under the target's differential harness out of 60.

Validity transfers to what shares the training vocabulary. At 360M, where the untrained model produces no valid program without the mask on any target, training lifts free validity on every held-out target. Above 360M the sign reverses on StableHLO Held-Out-200: the untrained models emit valid programs on 42 to 98% of prompts and their students on 14 to 44%. The full stack recovers all of that loss for SmolLM2 and under half of it for Gemma-1B. A full fine-tune at 1.7B reproduces it, 16.5 against the adapter's 14.0 and the untrained 42.0, so the loss is not an adapter artifact. The scf dialect, whose programs wrap the training dialects' arithmetic in regions, moves the other way at every scale, and correctness on it moves by at most 4 of 60. The student is therefore worse than the model it started from on one held-out dialect while better on another.

The mask does not repair what training removed. For Gemma-1B the student stays behind its own starting point on Held-Out-200 with the full stack restored, 48 against 96%, which is the part a fallback cannot fix: the mask is insurance against what training removed as well as against

what it never learned. The 3B student is 6 points behind on Spec-30, and no SmolLM2 cell is. Three explanations predict the drop. The weights may have learned the training dialects rather than validity as such (Section 4); a narrow fine-tune forgets whatever it does not rehearse (Chen et al., 2026); and the legal-mass term pushes down exactly the tokens a sibling dialect needs, since `stablehlo` operations lie outside the training grammar at every step. Because the same weights serve both dialects, pushing a token down where it is illegal also pushes it down where it is legal (Zabounidis et al., 2026). A control demotes the third: trained with no mask and no legal-mass term, the 1.7B GRPO student still collapses, to 8.5% free validity against the untrained 42.0. It is also the one the stack cannot rescue, 35.0 against 93 to 96%, because it never terminates inside the grammar on 130 of 200 prompts (Appendix B). Training under the mask leaves a student the mask can still repair off the training distribution; training without it does not. A KL penalty toward the base at 1.7B keeps StableHLO validity closer to the untrained model but leaves the syntax residual at 70 points: at this budget the displacement that buys validity on the training dialects is the one that costs it elsewhere, and the scf contrast shows the cost is selective by vocabulary. Verifier-trained constraint following is reported to transfer across classes of constraints (Wang et al., 2024); ours does not transfer across dialects.

A second training target replicates the decomposition. Repeating the 360M recipe on textual LLVM IR, with the template golds lowered by the compiler and `llvm-as` as the verifier, reproduces the whole pattern on a second target: validity moves into the weights, type domains internalize, and the gold-supervised student again leads on semantics (Table 11 in Appendix D).

7 Correctness, and the Price of Supplying It

Verifier acceptance is not correctness, and a schema can cost small models accuracy outright (Ray, 2026). Table 7 scores every cell with the gold-differential harness of Section 3, and Table 2 summarizes what each supervision source buys and costs at the two scales that have both an on-policy and a gold-supervised student; Figure 4 in Appendix D plots the two axes against each other.

weights	free valid %	R_1	correct free	correct full
360M untrained	0.0	–	0	26
360M SSD	69.5	15.0	21	20
360M SFT-gold	31.5	60.0	35	39
360M + one gold per group	35.0	48.5	36	42
1.7B untrained	19.0	–	16	48
1.7B SSD	57.5	20.5	46	43
1.7B SFT-gold	48.5	44.5	56	58
1.7B + one gold per group	55.5	33.0	57	57

Table 2: What each supervision source buys and what it costs, on arith+func (n=200; correctness of 143 gold-gate prompts, one greedy decode). R_1 is the syntax residual in pp. Differences under about 10 prompts lie inside the binomial interval (Section 8). Full per-stack cells are in Table 7 and the remaining ablations in Table 10.

After training at 360M the mask becomes functionally inert and correctness under it falls, from 26 of 143 for the untrained model to 20 for SSD in every seed. Off-policy supervision inverts the picture at both scales: it is the most correct student and the least valid one, and it internalizes the least syntax. Read against the binomial band of about 10 prompts (Section 8), no on-policy student differs from its base by more than the band at any scale, in either direction, and six on-policy cells sit above their base inside it. What carries the small-scale fall is repetition rather than any one interval: at 360M three SSD seeds give 20 and three RFT seeds 18 against a base of 26, and the sign repeats at 135M and Gemma-1B. SFT-gold is the only method that clears the band, at 360M, reaching its edge at 1.7B. Sampling resolves what one greedy decode leaves open, and it resolves it against a flat reading of the claim. Where the untrained model already samples correct programs, SSD raises sampled pass@1 by 6.8 and 9.1 points at 1.7B and by 12.1 at 3B, each with a paired bootstrap interval excluding zero, while RFT lowers it at both. Correctness does follow there. Reach does not: pass@8 does not rise at either scale, so the prompts eight samples reach beyond one collapse from 24 to between 5 and 11 (Table 15 in Appendix D). We read the two directions as one concentration

effect rather than a small-model artifact, an interpretation of the measurements and not a further measurement.

A semantics-carrying reward does not help. To test whether the gap is an artifact of a reward that grades only validity and interface, we add one differential trial per rollout against the template gold, worth as much as the whole task reward. At 360M it leaves correctness on the released pools where the task reward left it, within one prompt under every stack (Table 10 in Appendix D). The failure is not one of transfer: on the very pool the reward is computed against, the 360M student is no more correct than the task-reward one, while supervision on those golds reaches 72%. At 1.7B the reward does move that pool, by 17 points, and the released pool by 10, the width of the band (Appendix D). A reward selects among the model's own rollouts: it sharpens semantics the model already samples and does not introduce them where it does not. The contrast with verifier-rewarded rewriting at 3B (Fang et al., 2026) is the setting rather than the reward (Appendix D).

Supplying the program the model never samples. If the missing ingredient is the program rather than the reward, supplying it should help. Replacing the lowest-reward rollout of every group with the template gold, scored by the same reward inside the same group-relative advantage, recovers correctness at both scales and matches SFT-gold (Table 2), with the fewest interface mismatches of any run at either scale. At 360M the price is validity: free acceptance falls by 34.5 points and the syntax residual rises by 33.5, because the off-policy program pulls the student off its masked self, and two further seeds reproduce both sides (Table 9). The price looks smaller at 1.7B, but two controls dissolve most of that. Retraining the 360M student under the rank-32 adapters the 1.7B runs use lifts its free acceptance from 35.0 to 44.0%, and a second 1.7B seed lands at 44.0 against the first's 55.5 (Appendix D). Matched for adapter, the 360M value equals one of the two 1.7B seeds exactly, so we report the price as large under full fine-tuning and not separable from adapter capacity here. Sequencing the two signals does not escape the trade-off either (Appendix D).

The on-policy reference curve. Sampling the untrained 360M model under the mask gives what selection among its own rollouts can reach: 15 prompts in the first sample and 49 of 143 within 256, where the curve has all but flattened. The 360M students show the same signature, SSD raising pass@1 and lowering pass@16 while SFT-gold raises both (Table 14 in Appendix D). The per-prompt version is sharper, and it survives sampling both sides: giving each student 16 masked samples and the untrained model 256, SSD solves 27 prompts of which 1 lies outside the 49 the untrained model reaches, while SFT-gold solves 61 of which 26 lie outside, and one gold per group 61 of which 23 lie outside (Table 16 in Appendix D). Sixteen times more sampling does not bring the untrained model to the prompts gold supervision reaches. This deeper budget was affordable only at 360M. Training eight times longer does not change it either: free acceptance stays near 75% and full-stack correctness moves by 3 prompts of 143 (Appendix D). Distillation from a stronger teacher is already known to reach what reinforcement does not (Yue et al., 2025; Zhou, 2026); here the missing program is supplied inside the same objective and its price measured on an independent axis. The gap is not an interface convention either (Appendix D).

8 LIMITATIONS

Every cross-scale trend here is also a trend in the adapter: models up to 360M are fully fine-tuned, those above use rank-32 adapters at a five times larger learning rate. Two controls measure that confound rather than assume it away: a full fine-tune at 1.7B cuts the vocabulary residual to a quarter, and retraining the 360M gold-in-group ablation under adapters removes most of its apparent scale effect, which is why we claim none. The 3B rung is adapter-only. One greedy decode per prompt on 143 arith+func and 30 linalg prompts gives binomial intervals of roughly 10 and 5 prompts, so no linalg difference clears its interval and that column reports only that nothing collapsed. We claim only differences that exceed the interval, hold across seeds, or repeat across scales. The training pool is template-derived and the scope residual is a lower bound. Seeds are uneven: three at 360M for the main methods and the two load-bearing ablations, two at 1.7B, one elsewhere. The budget is small (2,000 prompts, four rollouts, two epochs), and eight times the steps moves free acceptance by one point and correctness by three prompts of 143, both inside their intervals.

AI USE STATEMENT

The research question, the hypotheses, the experimental design, and the analysis and interpretation of the results are the author's own work. Generative AI tools were used as a coding assistant only: they helped implement the training loop, the evaluation scripts, the validators for the new targets, and the figure and table generators, and all generated code was reviewed by the author and verified against unit tests and against the compiler's verifier. Generative AI tools were not used to generate datasets, to formulate hypotheses or claims, to design experiments, or to interpret results. The paper was written by the author; after the draft was complete, generative AI tools were used for language tuning and minor edits, which the author reviewed. The author takes full responsibility for all content of this submission, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

Every table names its frozen pool and n. Model checkpoints are pinned by repository revision, the compiler toolchain is a pinned LLVM 19.1.7 container, and the Python environment is a frozen lock file. Training and evaluation seeds, including the retry sampler's per-attempt seeds, are fixed in the scripts. The training pool, its datasheet and contamination report, the two new hand-authored test sets (Scf-Spec-60 and LLVM-Spec-60) with their references, every generation, and the trained checkpoints up to 3B parameters are part of the artifact. The artifact is at `https://anonymous.4open.science/r/slm-constraint-internalization-D8BE/`.

REFERENCES

Todd Zhou. When RLVR Shrinks the Reasoning Boundary: Diagnosing Pass@k Inversion. *arXiv preprint arXiv:2607.20543*, 2026.

Matteo Biagiola, Jahrim Gabriele Cesario, Luca Di Grazia, George Zakhour, Guido Salvaneschi. The Alignment Problem in Constrained Code Generation. *arXiv preprint arXiv:2606.21619*, 2026.

Andrei Liviu Nicolicioiu, Mohammad Pezeshki, Aaron Courville. On-Policy Self-Distillation with Sampled Demonstrations Reduces Output Diversity. *arXiv preprint arXiv:2606.26091*, 2026.

Igor Lima Strozzi. Teacher-Free Self-Training Amplifies but Does Not Compound: A Pass@K Crossover on a Free-Verifier Domain. *arXiv preprint arXiv:2606.07856*, 2026.

Jaideep Ray. The Constraint Tax: Measuring Validity-Correctness Tradeoffs in Structured Outputs for Small Language Models. *arXiv preprint arXiv:2605.26128*, 2026.

Vinayshekhar Bannihatti Kumar, Disha Makhija, Manoj Ghuhan Arivazhagan, Rashmi Gangadharaiah. Syntax Without Semantics: Teaching Large Language Models to Code in an Unseen Language. *Conference on Language Modeling (COLM)*, 2026. arXiv:2605.15607.

Plawan Kumar Rath. Cross-Dialect Generalization Without Retraining: Benchmarks and Evaluation of Schema-Derived Constrained Decoding for MLIR. *arXiv preprint arXiv:2607.18254*, 2026.

Egor Skopin, Evgeny Kotelnikov. Improving Small Language Models for Code Generation with Reinforcement Learning from Verification Feedback. *Artificial Intelligence and Natural Language (AINL)*, 2026. arXiv:2605.30478.

Yihong Dong, Jianha Xiao, Xue Jiang, Xuyuan Guo, Zhiyuan Fan, Jiaru Qian, Kechi Zhang, Jia Li, Zhi Jin, Ge Li. Evaluating the Formal Reasoning Capabilities of Large Language Models through Chomsky Hierarchy. *arXiv preprint arXiv:2604.02709*, 2026.

Aryan Sharma, Cutter Dawes, Shivam Raval. Dissociating Decodability and Causal Use in Bracket-Sequence Transformers. *arXiv preprint arXiv:2604.22128*, 2026.

Yuming Yan, Kai Tang, Sihong Chen, Ke Xu, Dan Hu, Qun Yu, Pengfei Hu. S-GRPO: Unified Post-Training for Large Vision-Language Models. *arXiv preprint arXiv:2604.16557*, 2026.

Ruixiang Zhang, Richard He Bai, Huangjie Zheng, Navdeep Jaitly, Ronan Collobert, Yizhe Zhang. Embarrassingly Simple Self-Distillation Improves Code Generation. *arXiv preprint arXiv:2604.01193*, 2026.

Xiangxin Fang, Jiaqin Kang, Rodrigo C. O. Rocha, Sam Ainsworth, Lev Mukhanov. LLM-VeriOpt: Verification-Guided Reinforcement Learning for LLM-Based Compiler Optimization. *IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2026.

Valerio Belcamino, Nicholas Attolino, Alessio Capitanelli, Fulvio Mastrogiovanni. On the Generalization Gap in LLM Planning: Tests and Verifier-Reward RL. *arXiv preprint arXiv:2601.14456*, 2026.

Aleksander Boruch-Gruszecki, Yangtian Zi, Zixuan Wu, Tejas Oberoi, Carolyn Anderson, Joydeep Biswas, Arjun Guha. Agnostics. *International Conference on Learning Representations (2026)*, 2026. arXiv:2508.04865.

Fan Chen, Audrey Huang, Noah Golowich, Sadhika Malladi, Adam Block, Jordan T. Ash, Akshay Krishnamurthy, Dylan J. Foster. The Coverage Principle: How Pre-Training Enables Post-Training. *International Conference on Learning Representations (2026)*, 2026. arXiv:2510.15020.

Howard Chen, Noam Razin, Karthik Narasimhan, Danqi Chen. Retaining by Doing: The Role of On-Policy Data in Mitigating Forgetting. *International Conference on Machine Learning (2026)*, 2026. arXiv:2510.18874.

Bishwamittra Ghosh, Soumi Das, Till Speicher, Qinyuan Wu, Mohammad Aflah Khan, Deepak Garg, Krishna P. Gummadi, Evimaria Terzi. Fine-tuning vs. In-context Learning in Large Language Models: A Formal Language Learning Perspective. *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, 2026. arXiv:2604.23267.

Chuxuan Hu, Yuxuan Zhu, Antony Kellermann, Caleb Biddulph, Suppakit Waiwitlikhit, Jason Benn, Daniel Kang. Breaking Barriers: Do Reinforcement Post Training Gains Transfer To Unseen Domains?. *International Conference on Learning Representations (2026)*, 2026. arXiv:2506.19733.

Po-Nien Kung, Zhen Yang, Jeffrey Luo, Cheng-Fu Yang, Haikang Deng, Zi-Yi Dou, Yinfei Yang, Nanyun Peng, Zhe Gan, Kai-Wei Chang. Learning Structured Reasoning via Tractable Trajectory Control. *International Conference on Machine Learning (2026)*, 2026. arXiv:2603.01641.

Lu Ma, Hao Liang, Meiyi Qiang, Lexiang Tang, Xiaochen Ma, Zhen Hao Wong, Junbo Niu, Chengyu Shen, Runming He, Yanhao Li, Bin Cui, Wentao Zhang. Learning What Reinforcement Learning Can't: Interleaved Online Fine-Tuning for Hardest Questions. *International Conference on Learning Representations (2026)*, 2026. arXiv:2506.07527.

Kohsei Matsutani, Shota Takashiro, Gouki Minegishi, Takeshi Kojima, Yusuke Iwasawa, Yutaka Matsuo. RL Squeezes, SFT Expands: A Comparative Study of Reasoning LLMs. *International Conference on Learning Representations (2026)*, 2026. arXiv:2509.21128.

Avinash Reddy, Thayne Walker, Jaime Ide, Amrit Singh Bedi. The Hidden Cost of Structured Generation in LLMs: Draft-Conditioned Constrained Decoding. *International Conference on Machine Learning (2026)*, 2026. arXiv:2603.03305.

Idan Shenfeld, Jyothish Pari, Pulkit Agrawal. RL's Razor: Why Online Reinforcement Learning Forgets Less. *International Conference on Learning Representations (2026)*, 2026. arXiv:2509.04259.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, Pulkit Agrawal. Self-Distillation Enables Continual Learning. *International Conference on Machine Learning (2026)*, 2026. arXiv:2601.19897.

Jiin Woo, Shaowei Zhu, Allen Nie, Zhen Jia, Yida Wang, Youngsuk Park. TritonRL: Training LLMs to Think and Code Triton Without Cheating. *Conference on Language Modeling (COLM)*, 2026. arXiv:2510.17891.

Renos Zabounidis, Roy Siegelmann, Mohamad Qadri, Woojun Kim, Simon Stepputtis, Katia P. Sycara. Overcoming Valid Action Suppression in Unmasked Policy Gradient Algorithms. *arXiv preprint arXiv:2603.09090*, 2026.

Wenhao Zhang, Yuexiang Xie, Yuchang Sun, Yanxi Chen, Guoyin Wang, Yaliang Li, Bolin Ding, Jingren Zhou. On-Policy RL Meets Off-Policy Experts: Harmonizing Supervised Fine-Tuning and Reinforcement Learning via Dynamic Weighting. *International Conference on Learning Representations (2026)*, 2026. arXiv:2508.11408.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, Aditya Grover. Self-Distilled Reasoner: On-Policy Self-Distillation for Large Language Models. *International Conference on Machine Learning (2026)*, 2026. arXiv:2601.18734.

Sairam Vaidya, Marcel Böhme, Loris D'Antoni. Bootstrapping Fuzzers for Compilers of Low-Resource Language Dialects Using Language Models. *arXiv preprint arXiv:2512.05887*, 2025.

Francesca Lucchetti, Arjun Guha. Understanding How CodeLLMs (Mis)Predict Types with Activation Steering. *Proceedings of the 8th BlackboxNLP Workshop*, pp. 358–397, 2025. arXiv:2404.01903.

Xinting Huang, Andy Yang, Satwik Bhattamishra, Yash Sarrof, Andreas Krebs, Hattie Zhou, Preetum Nakkiran, Michael Hahn. A Formal Framework for Understanding Length Generalization in Transformers. *International Conference on Learning Representations*, 2025. arXiv:2410.02140.

Zhechong Huang, Zhao Zhang, Ruyi Ji, Tingxuan Xia, Qihao Zhu, Qinxiang Cao, Zeyu Sun, Wiggin Zhou, Yingfei Xiong. Learning to Guarantee Type Correctness in Code Generation through Type-Guided Program Synthesis. *arXiv preprint arXiv:2510.10216*, 2025.

Jatin Prakash, Anirudh Buvanesh. What Can You Do When You Have Zero Rewards During RL?. *arXiv preprint arXiv:2510.03971*, 2025.

Hao Wu, Wei Liu. GCPO: When Contrast Fails, Go Gold. *arXiv preprint arXiv:2510.07790*, 2025.

Maximilian Schall, Gerard de Melo. The Hidden Cost of Structure: How Constrained Decoding Affects Language Model Performance. *Recent Advances in Natural Language Processing (RANLP)*, 2025.

Elie Bakouch, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Lewis Tunstall, Carlos Miguel Patiño, Edward Beeching, Aymeric Roucher, others. SmolLM3: Smol, Multilingual, Long-Context Reasoner. Hugging Face blog, `https://huggingface.co/blog/smollm3`, 2025.

Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, Yejin Choi. The Invisible Leash: Why RLVR May or May Not Escape Its Origin. *arXiv preprint arXiv:2507.14843*, 2025.

Andre Wang He, Daniel Fried, Sean Welleck. Rewarding the Unlikely: Lifting GRPO Beyond Distribution Sharpening. *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 25548–25560, 2025. arXiv:2506.02355.

Niels Mündler, Jingxuan He, Hao Wang, Koushik Sen, Dawn Song, Martin Vechev. Type-Constrained Code Generation with Language Models. *Proceedings of the ACM on Programming Languages (PLDI)*, 9, 2025.

Minwu Kim, Anubhav Shrestha, Safal Shrestha, Aadim Nepal, Keith Ross. Reinforcement Learning vs. Distillation: Understanding Accuracy and Capability in LLM Reasoning. *arXiv preprint arXiv:2505.14216*, 2025.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, Yi Dong. ProRL: Prolonged Reinforcement Learning Expands Reasoning Boundaries in Large Language Models. *Advances in Neural Information Processing Systems*, 2025. arXiv:2505.24864.

Yiting Wang, Guoheng Sun, Wanghao Ye, Gang Qu, Ang Li. VeriReason: Reinforcement Learning with Testbench Feedback for Reasoning-Enhanced Verilog Generation. *arXiv preprint arXiv:2505.11849*, 2025.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, Yue Zhang. Learning to Reason under Off-Policy Guidance. *Advances in Neural Information Processing Systems*, 2025. arXiv:2504.14945.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, Gao Huang. Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model?. *Advances in Neural Information Processing Systems*, 38, 2025. arXiv:2504.13837.

Rosie Zhao, Alexandru Meterez, Sham Kakade, Cengiz Pehlevan, Samy Jelassi, Eran Malach. Echo Chamber: RL Post-training Amplifies Behaviors Learned in Pretraining. *Conference on Language Modeling (COLM)*, 2025. arXiv:2504.07912.

Gemma Team. Gemma 3 Technical Report. *arXiv preprint arXiv:2503.19786*, 2025.

Saibo Geng, Hudson Cooper, Michał Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, Harsha Nori. JSONSchemaBench: A Rigorous Benchmark of Structured Outputs for Language Models. *arXiv preprint arXiv:2501.10868*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, others. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. Preprint arXiv:2501.12948.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, others. SmolLM2. *Conference on Language Modeling (COLM)*, 2025. arXiv:2502.02737.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ziyi Xu, Yilong Zhao, Ruihang Lai, Tianqi Chen. XGrammar: Flexible and Efficient Structured Generation Engine for Large Language Models. *Proceedings of Machine Learning and Systems*, 7, 2025. arXiv:2411.15100.

Audrey Huang, Adam Block, Dylan J. Foster, Dhruv Rohatgi, Cyril Zhang, Max Simchowitz, Jordan T. Ash, Akshay Krishnamurthy. Self-Improvement in Language Models: The Sharpening Mechanism. *International Conference on Learning Representations (2025)*, 2025. arXiv:2412.01951.

Hailong Jiang, Jianfeng Zhu, Yao Wan, Bo Fang, Hongyu Zhang, Ruoming Jin, Qiang Guan. Can Large Language Models Understand Intermediate Representations in Compilers?. *Proceedings of the 42nd International Conference on Machine Learning*, 267:27851–27872, 2025. arXiv:2502.06854.

Kevin Lu, Thinking Machines Lab. On-Policy Distillation. Thinking Machines Lab: Connectionism, 2025. DOI 10.64434/tml.20251026.

Yaxi Lu, Haolun Li, Xin Cong, Zhong Zhang, Yesai Wu, Yankai Lin, Zhiyuan Liu, Fangming Liu, Maosong Sun. Learning to Generate Structured Output with Schema Reinforcement Learning. *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4905–4918, 2025.

Hadas Orgad, Michael Toker, Zorik Gekhman, Roi Reichart, Idan Szpektor, Hadas Kotek, Yonatan Belinkov. LLMs Know More Than They Show: On the Intrinsic Representation of LLM Hallucinations. *International Conference on Learning Representations (2025)*, 2025. arXiv:2410.02707.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, Gagandeep Singh. SynCode. *Transactions on Machine Learning Research*, 2025. arXiv:2403.01632.

Yiwei Wu, Atticus Geiger, Raphaël Millière. How Do Transformers Learn Variable Binding in Symbolic Programs?. *Proceedings of the 42nd International Conference on Machine Learning*, 267:67284–67299, 2025. arXiv:2505.20896.

Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, Loris D'Antoni. Grammar-Aligned Decoding. *Advances in Neural Information Processing Systems*, 37, 2024.

Edward J. Hu, Moksh Jain, Eric Elmoznino, Younesse Kaddar, Guillaume Lajoie, Yoshua Bengio, Nikolay Malkin. Amortizing Intractable Inference in Large Language Models. *International Conference on Learning Representations*, 2024. arXiv:2310.04363.

Luca Beurer-Kellner, Marc Fischer, Martin Vechev. Guiding LLMs the Right Way: Fast, Non-Invasive Constrained Generation. *Proceedings of the 41st International Conference on Machine Learning*, pp. 3658–3673, 2024.

Nadav Borenstein, Anej Svete, Robin Chan, Josef Valvoda, Franz Nowak, Isabelle Augenstein, Eleanor Chodroff, Ryan Cotterell. What Languages are Easy to Language-Model? A Perspective from Learning Probabilistic Regular Languages. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. arXiv:2406.04289.

Chris Cummins, Volker Seeker, Dejan Grubisic, Baptiste Roziere, Jonas Gehring, Gabriel Synnaeve, Hugh Leather. Meta Large Language Model Compiler: Foundation Models of Compiler Optimization. *arXiv preprint arXiv:2407.02524*, 2024.

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, Olivier Bachem. On-Policy Distillation of Language Models: Learning from Self-Generated Mistakes. *International Conference on Learning Representations*, 2024.

Yuxian Gu, Li Dong, Furu Wei, Minlie Huang. MiniLLM: Knowledge Distillation of Large Language Models. *International Conference on Learning Representations*, 2024.

Charles Jin, Martin Rinard. Emergent Representations of Program Semantics in Language Models Trained on Programs. *Proceedings of the 41st International Conference on Machine Learning*, 2024. arXiv:2305.11169.

Stefan Heimersheim, Neel Nanda. How to Use and Interpret Activation Patching. *arXiv preprint arXiv:2404.15255*, 2024.

Indraneil Paul, Goran Glavaš, Iryna Gurevych. IRCoder: Intermediate Representations Make Language Models Robust Multilingual Code Generators. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 15023–15041, 2024. arXiv:2403.03894.

Fei Wang, Chao Shang, Sarthak Jain, Shuai Wang, Qiang Ning, Bonan Min, Vittorio Castelli, Yassine Benajiba, Dan Roth. From Instructions to Constraints: Language Model Alignment with Automatic Constraint Verification. *arXiv preprint arXiv:2403.06326*, 2024.

Nikhil Prakash, Tamar Rott Shaham, Tal Haklay, Yonatan Belinkov, David Bau. Fine-Tuning Enhances Existing Mechanisms: A Case Study on Entity Tracking. *International Conference on Learning Representations*, 2024. arXiv:2402.14811.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, Daya Guo. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300*, 2024.

Samyak Jain, Robert Kirk, Ekdeep Singh Lubana, Robert P. Dick, Hidenori Tanaka, Edward Grefenstette, Tim Rocktäschel, David Scott Krueger. Mechanistically Analyzing the Effects of Fine-Tuning on Procedurally Defined Tasks. *International Conference on Learning Representations (2024)*, 2024. arXiv:2311.12786.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Joshua Susskind, Samy Bengio, Preetum Nakkiran. What Algorithms can Transformers Learn? A Study in Length Generalization. *International Conference on Learning Representations (2024)*, 2024. arXiv:2310.16028.

Kareem Ahmed, Kai-Wei Chang, Guy Van den Broeck. A Pseudo-Semantic Loss for Autoregressive Models with Logical Constraints. *Advances in Neural Information Processing Systems*, 36, 2023. arXiv:2312.03905.

Saibo Geng, Martin Josifoski, Maxime Peyrard, Robert West. Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10932–10952, 2023.

Kexun Zhang, Hongqiao Chen, Lei Li, William Wang. Don't Fine-Tune, Decode: Syntax Error-Free Tool Use via Constrained Decoding. *arXiv preprint arXiv:2310.07075*, 2023.

Chris Cummins, Volker Seeker, Dejan Grubisic, Mostafa Elhoushi, Youwei Liang, Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Kim Hazelwood, Gabriel Synnaeve, Hugh Leather. Large Language Models for Compiler Optimization. *arXiv preprint arXiv:2309.07062*, 2023.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, Jingren Zhou. Scaling Relationship on Learning Mathematical Reasoning with Large Language Models. *arXiv preprint arXiv:2308.01825*, 2023.

Brandon T. Willard, Rémi Louf. Efficient Guided Generation for Large Language Models. *arXiv preprint arXiv:2307.09702*, 2023.

Subhro Roy, Sam Thomson, Tongfei Chen, Richard Shin, Adam Pauls, Jason Eisner, Benjamin Van Durme. BenchCLAMP: A Benchmark for Evaluating Language Models on Syntactic and Semantic Parsing. *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2023. arXiv:2206.10668.

Grégoire Delétang, Anian Ruoss, Jordi Grau-Moya, Tim Genewein, Li Kevin Wenliang, Elliot Catt, Chris Cundy, Marcus Hutter, Shane Legg, Joel Veness, Pedro A. Ortega. Neural Networks and the Chomsky Hierarchy. *International Conference on Learning Representations*, 2023.

Chandra Bhagavatula, Jena D. Hwang, Doug Downey, Ronan Le Bras, Ximing Lu, Lianhui Qin, Keisuke Sakaguchi, Swabha Swayamdipta, Peter West, Yejin Choi. I2D2: Inductive Knowledge Distillation with NeuroLogic and Self-Imitation. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9614–9630, 2023. arXiv:2212.09246.

Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, Yoon Kim. Grammar Prompting for Domain-Specific Language Generation with Large Language Models. *Advances in Neural Information Processing Systems*, 36, 2023. arXiv:2305.19234.

Kevin Meng, David Bau, Alex Andonian, Yonatan Belinkov. Locating and Editing Factual Associations in GPT. *Advances in Neural Information Processing Systems*, 35, 2022.

Eric Zelikman, Yuhuai Wu, Jesse Mu, Noah D. Goodman. STaR: Bootstrapping Reasoning With Reasoning. *Advances in Neural Information Processing Systems*, 35, 2022.

Charlie Snell, Dan Klein, Ruiqi Zhong. Learning by Distilling Context. *arXiv preprint arXiv:2209.15189*, 2022.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. *International Conference on Learning Representations*, 2022.

Gabriel Poesia, Oleksandr Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, Sumit Gulwani. Synchromesh: Reliable Code Generation from Pre-trained Language Models. *International Conference on Learning Representations*, 2022.

Yonatan Belinkov. Probing Classifiers: Promises, Shortcomings, and Advances. *Computational Linguistics*, 48(1), 2022. arXiv:2102.12452.

Xin Wang, Yasheng Wang, Yao Wan, Fei Mi, Yitong Li, Pingyi Zhou, Jin Liu, Hao Wu, Xin Jiang, Qun Liu. Compilable Neural Code Generation with Compiler Feedback. *Findings of the Association for Computational Linguistics: ACL 2022*, 2022. arXiv:2203.05132.

Sergey Troshin, Nadezhda Chirkova. Probing Pretrained Models of Source Codes. *Proceedings of the Fifth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pp. 371–383, 2022. arXiv:2202.08975.

Shengyi Huang, Santiago Ontañón. A Closer Look at Invalid Action Masking in Policy Gradient Algorithms. *International FLAIRS Conference Proceedings*, 35, 2022. arXiv:2006.14171.

Torsten Scholak, Nathan Schucher, Dzmitry Bahdanau. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

Shunyu Yao, Binghui Peng, Christos Papadimitriou, Karthik Narasimhan. Self-Attention Networks Can Process Bounded Hierarchical Languages. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 3770–3785, 2021.

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

Nuno P. Lopes, Juneyoung Lee, Chung-Kil Hur, Zhengyang Liu, John Regehr. Alive2: Bounded Translation Validation for LLVM. *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI)*, pp. 65–79, 2021.

Emiel van Miltenburg, Chris van der Lee, Emiel Krahmer. Preregistering NLP Research. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics*, 2021. arXiv:2103.06944.

Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, Oleksandr Zinenko. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 2–14, 2021.

John Hewitt, Percy Liang. Designing and Interpreting Probes with Control Tasks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, 2019. arXiv:1909.03368.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, Guy Van den Broeck. A Semantic Loss Function for Deep Learning with Symbolic Knowledge. *Proceedings of the 35th International Conference on Machine Learning*, 2018. arXiv:1711.11157.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, Eric Xing. Harnessing Deep Neural Networks with Logic Rules. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016. arXiv:1603.06318.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv preprint arXiv:1503.02531*, 2015. NIPS 2014 Deep Learning Workshop.

Xuejun Yang, Yang Chen, Eric Eide, John Regehr. Finding and Understanding Bugs in C Compilers. *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 283–294, 2011.

A Training pool datasheet and contamination report

The training pool is generated by templates rather than mined from existing test suites or repositories, so that it can be made disjoint from every released evaluation pool; the released pools of Rath (2026) are used only as frozen test sets. Templates in the artifact's data module produce natural-language requests and their gold MLIR programs for seven arith and func families (binary operations, chains of two to five operations, constants, comparisons and selects, casts, three-way minimum and maximum, and memref load, store and dim) and for linalg over the 12 named operations the grammar admits; the kept pool carries 16 arith and func and 7 linalg family labels, since a family such as chains splits by length. Each family draws operand types, shapes, and literals from bounded slot ranges, and every request is phrased with one of eight prefix variants. Generation oversamples three times and then applies three gates: the row must parse under the parse grammar, be replayable under the C1+C2 generation grammar, and pass `mlir-opt` verification with LLVM 19.1.7. All 12,242 rows that survive deduplication pass all three gates.

Disjointness. Candidates are checked against 790 protected records: the released benchmarks used in this paper and in Rath (2026), the held-out and out-of-grammar StableHLO pools, the functional references, and both frozen evaluation pools. A candidate is dropped when its gold is alpha-equivalent to a protected gold after SSA and symbol names are canonicalized (2,955 drops), when its normalized request shares any 8-gram with a protected request (3,592 drops), or when its request duplicates another candidate (5,211 drops). The kept pool has 7,257 training rows (6,000 arith and func, 1,257 linalg) and 500 dev rows. Training uses the first 2,000 training prompts and the first 100 dev rows.

Known skew. The linalg template space is small. Only 1,382 unique linalg rows survive the gates, and the matmul and unary elementwise families are under-represented because their canonical forms coincide with benchmark golds and were dropped. The requests are template phrased. The coverage pool (v1c) used in Section 4 adds fourteen template families for the concepts on which the students invent operations (allocation and deallocation, cast chains, clamps, polynomials, negation, float minimum and maximum, bit casts, conditional selects, comparisons with constants, and two-input elementwise linalg), built with the same generator seed, oversampling, gates, and disjointness filter; new families make up 28% of its 7,239 training rows. Golds are used only by the SFT-gold baseline and for deduplication. The pool ships with the artifact under Apache-2.0.

B CONSTRAINT LAYER DEFINITIONS AND VALIDATORS

Distance from the base. On-policy updates reportedly forget less, by a law tying forgetting to how far a run moves on its own task (Shenfeld et al., 2026). Our 1.7B runs differ fourfold in per-token KL to the untrained base, but transfer is measured for the self-distillation arm only, so we report the distance without testing the relation.

Non-termination under the mask. The unmasked GRPO student at 1.7B fails to emit an end state on 130 of 200 StableHLO Held-Out-200 prompts under C1 and above, running to the token cap and averaging 1072 tokens against 126 for the mask-trained student. The count is identical at a 600-token and a 1600-token cap, so it is non-termination rather than truncation.

Memorization check. Mean maximum 8-gram overlap between a trained cell's held-out-target outputs and the training golds is at most 0.11 in every trained cell, so the transfer numbers are not verbatim copying.

The generation grammar. Both context-free layers are one Lark grammar compiled by llguidance into a token mask. The grammar bounds identifier length and body size, so the language it samples is finite; the claim that SSA scope is not context-free, and therefore needs a validator over the finished program rather than a token mask, is a claim about the unbounded constraint. The grammar is a strict subset of the parse grammar, uses explicit whitespace productions, bounds identifiers (SSA names to 32 characters, symbols to 48), and caps a function body at six operations by explicit alternation so that decoding terminates. The C1-only variant admits every operation with any type; the C1+C2 variant splits each family by type domain, so integer arithmetic takes only integer types, float arithmetic only float types, comparison predicates match their operand class, constant literals match their type, the five cast families respect their source and target domains, and `return` is either void or typed. The 22 operation alternatives cover arith, func, memref, and the 12 named linalg operations.

The scope validator (C3). The scope layer is a post-hoc validator applied as rejection sampling with five retries. It keeps a per-function symbol table seeded from the function parameters, adds a typed definition for every `%name = op ... : T` line, and checks every use against the type obligation of the operation pattern it appears in (for example, both operands of an integer binary operation carry the trailing type, a comparison returns `i1`, a load's indices are `index` and its result is the memref element type). Use before definition and type mismatch are both violations. Lines whose operation the validator does not recognize are abstained on: no definition is added and no use is checked, so the validator can miss a violation but never rejects a valid program. Offline, the same validator provides the per-token oracle used to build the dense scope mask of the ablations.

Shape and interface diagnostics (C4). No mask or validator implements this layer, whose conditions depend on values a decoder cannot see. Its violation rate is the fraction of parse-valid programs on which `mlir-opt` reports a shape, rank, or element-type diagnostic, and the functional harness separately reports interface mismatches, that is verifier-accepted programs whose function signature differs from the gold.

New targets. The portability experiments reuse the same construction. LLVM IR has a C1 and a C1+C2 grammar and a scope validator over its typed instructions; the scf and StableHLO dialects have region-aware scope validators that track definitions across nested blocks. The scf and LLVM IR targets each ship with a hand-authored 60-prompt specification pool and its references; StableHLO uses the released Spec-30 and Held-Out-200 sets.

C Training details

The collapse under a verifier-only reward. The first run used the verifier alone as the reward. It reached 100% dev verify-rate and then emitted the same program for all 125 benchmark prompts: the constant-returning function from the prompt's few-shot, followed by newlines to the token limit. A prompt-independent reward has a prompt-independent optimum, and the legal-mass term does not oppose it because that program is legal. The reward is therefore $r = \text{verify} \cdot \text{sig}(x, g) \cdot \text{J}(x, g)$, where sig is the graded agreement between the generated and gold function signatures and J is the Jaccard similarity of operation multisets; the trivial program scores zero on every prompt. Rollouts that finish receive an explicit end-of-sequence token, and the replay allows only that token at the accepting state, so the loss also teaches termination. Dev evaluation reports the number of distinct programs as a collapse detector. The detector fires on the Gemma family: Gemma-1B under RFT keeps few rollouts per batch and collapses even with the kept-fraction step scaling, and Gemma-270M collapses under both SSD and RFT by step 25 (dev verify 1.0 with 3 to 7 distinct programs and a task reward of 0.03), so Gemma-270M appears only in the zero-training ladder and the family control rests on Gemma-1B.

Hyperparameters. Every run sees the same 2,000 training prompts for two epochs in batches of 32 prompts with four rollouts per prompt (126 optimizer steps), samples at temperature 0.8 with at most 300 new tokens, and uses $\beta = 0.1$ for the frozen masked teacher and $\lambda = 1$ for the task reward. Models up to 360M are fully fine-tuned with learning rate 2×10^{-5}; models of 1B and above train LoRA adapters of rank 32 with learning rate 10^{-4} and adapter scale 2 (scale 20 degraded SSD and collapsed RFT at 1.7B). Loss and teacher forwards run in micro-batches of 8 sequences, or 4 for the 3B model and for Gemma, whose 262k vocabulary widens the logits. RFT scales its step by the fraction of kept rollouts so that a batch with few accepted samples does not take a full-size step.

The legal-mass loss. With A_t the allowed set after the sampled prefix,

$$\mathcal{L}_{\text{mask}}(\theta) = \mathbb{E}_{x \sim \pi_\theta^{\text{mask}}} \Big[\sum_t - \log \sum_{v \in A_t} \pi_\theta(v \mid x_{<t}) \Big], \tag{1}$$

which is the on-policy-distillation implicit reward with the teacher set to the masked self: for every sampled token, $\log \pi^{\text{mask}}(x_t) - \log \pi_\theta(x_t) = -\log Z_t$.

Checkpoint selection. Every 25 steps the first 100 dev-pool prompts are decoded greedily without the mask, and the checkpoint with the highest free-decoding task reward is kept. Table 4 lists the selected step and wall-clock for the seed-0 runs.

D Additional results

Our zero-training ladder does not reproduce the published one on arith+func. Rath (2026) report 34.5, 54.5, 54.5 and 67.5% for the 1.7B model across the four stacks, where we measure 19.0, 47.0, 48.5 and 57.5; on linalg they report 49.6, 68.0, 72.0 and 72.8 against our 53.6, 71.2, 78.4 and 79.2, which is close. We use the released pools and the pinned model revision but our own harness, prompt assembly and verifier build, and we have no decisive explanation for the arith+func

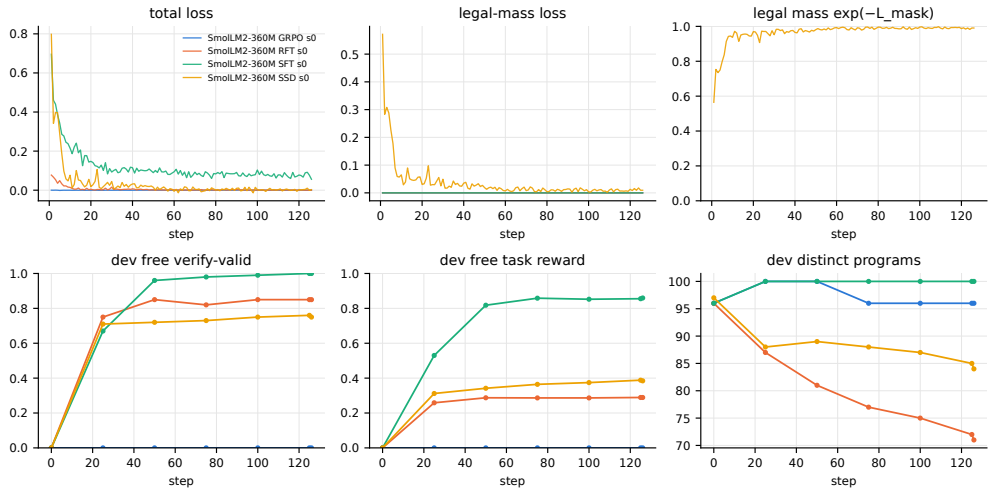


Figure 3: Training curves for the 360M runs: loss, legal mass, and dev free-decoding verify rate, task reward, and distinct-program count per evaluation.

	arith+func (n=200)				linalg (n=125)			
model	free	C1	C1+C2	C1+C2+C3	free	C1	C1+C2	C1+C2+C3
SmolLM2-135M	0.0	57.5	57.5	65.0	0.0	16.8	20.0	48.8
SmolLM2-360M	0.0	39.5	40.5	58.5	0.0	15.2	54.4	71.2
SmolLM2-1.7B	19.0	47.0	48.5	57.5	53.6	71.2	78.4	79.2
SmolLM3-3B	14.0	45.5	47.5	56.0	5.6	12.0	38.4	55.2
Gemma-3-270M	48.5	83.5	83.5	89.5	6.4	16.8	39.2	63.2
Gemma-3-1B	16.0	38.5	38.5	41.5	12.0	70.4	77.6	80.0

Table 3: Zero-training constraint ladder, verify-valid rate (%) under each constraint stack, on the two frozen pools .

gap. Every comparison in this paper is therefore against our own ladder, and the prior study's pools rather than its numbers are the continuity anchor.

Table 4: Selected checkpoint and wall-clock per seed-0 run on one Apple M3 Ultra. Steps run to 126; the selected step is the last dev evaluation whose free-decoding reward was highest.

Model	Method	Adapter	Selected step	Wall-clock (h)
SmolLM2-135M	SSD	full	125	0.8
SmolLM2-135M	RFT	full	126	0.6
SmolLM2-360M	SSD	full	125	2.6
SmolLM2-360M	RFT	full	126	0.8
SmolLM2-360M	GRPO	full	126	1.5
SmolLM2-360M	SFT-gold	full	126	0.2
SmolLM2-1.7B	SSD	LoRA	100	3.3
SmolLM2-1.7B	RFT	LoRA	50	2.3
SmolLM3-3B	SSD	LoRA	126	4.5
SmolLM3-3B	RFT	LoRA	75	3.2
Gemma-3-1B	SSD	LoRA	50	2.5
Gemma-3-1B	RFT	LoRA	25	0.8

Pre-registration. The thresholds of Table 5 were fixed before the trained checkpoints were evaluated, in the sense of van Miltenburg et al. (2021), and are reported here with their outcome whether or not they were met.

model	pool	R_1	R_2	R_3	R_4
G-1B SSD s0	arith	+8.5	+0.0 ✓	+6.0	+0.0 ✓
G-1B SSD s0	linalg	+64.0	+0.0 ✓	+0.0 ✓	-10.4
1.7B SSD s0	arith	+20.5	-2.5	-3.5 ✓	-1.0 ✓
1.7B SSD s0	linalg	+51.2	+0.0 ✓	-1.6 ✓	-11.2
360M SSD s0	arith	+15.0	+1.0 ✓	+4.5	-1.5 ✓
360M SSD s0	linalg	+43.2	+0.0 ✓	+0.8 ✓	-9.6
3B SSD s0	arith	+31.5	-3.0	-3.0 ✓	-2.0
3B SSD s0	linalg	+78.4	+0.0 ✓	+0.0 ✓	-22.4

Table 5: Framing gate: per-layer residual (pp) after SSD at every scale; ✓marks an internalized layer (95% CI contains 0 and point estimate $\leq$ 2pp).

checkpoint	pool	R_1 syntax	R_2 type	R_3 scope	R_4 shape
G-1B RFT s0	arith+func	+85.0 [+80, +90]	+0.0 [+0, +0]	+0.0 [+0, +0]	+0.0 [+0, +0]
G-1B RFT s0	linalg	+0.0 [+0, +0]	+0.0 [+0, +0]	-0.8 [-5, +3]	+0.0 [+0, +0]
G-1B SSD s0	arith+func	+8.5 [+5, +12]	+0.0 [+0, +0]	+6.0 [+2, +10]	+0.0 [+0, +0]
G-1B SSD s0	linalg	+64.0 [+55, +72]	+0.0 [+0, +0]	+0.0 [-2, +2]	-10.4 [-16, -6]
1.7B GRPO s0	arith+func	+59.5 [+52, +66]	-5.0 [-8, -2]	-16.5 [-22, -10]	-2.0 [-4, -0]
1.7B GRPO s0	linalg	+53.6 [+45, +62]	+0.0 [+0, +0]	+0.8 [-2, +3]	-11.2 [-17, -6]
1.7B RFT s0	arith+func	+30.5 [+24, +37]	+0.0 [+0, +0]	-9.0 [-14, -4]	+0.0 [+0, +0]
1.7B RFT s0	linalg	+51.2 [+42, +60]	+9.6 [+5, +15]	-5.6 [-10, -2]	-17.6 [-24, -11]
1.7B SFT s0	arith+func	+44.5 [+38, +52]	-1.5 [-4, +0]	-24.5 [-30, -18]	-3.5 [-6, -1]
1.7B SFT s0	linalg	+60.0 [+51, +68]	+0.0 [+0, +0]	+12.0 [+6, +18]	-17.6 [-24, -11]
1.7B SSD s0	arith+func	+20.5 [+15, +26]	-2.5 [-5, -0]	-3.5 [-8, +1]	-1.0 [-2, +0]
1.7B SSD s0	linalg	+51.2 [+42, +60]	+0.0 [+0, +0]	-1.6 [-4, +0]	-11.2 [-18, -6]
135M RFT s0	arith+func	+41.0 [+34, +48]	+0.0 [+0, +0]	-3.5 [-6, -1]	-2.5 [-5, -0]
135M RFT s0	linalg	+44.0 [+35, +53]	+0.8 [+0, +2]	-4.8 [-9, -2]	-12.8 [-19, -7]
135M SSD s0	arith+func	+2.0 [+0, +4]	+2.0 [+0, +4]	+9.0 [+6, +13]	+0.0 [+0, +0]
135M SSD s0	linalg	+42.4 [+34, +51]	+1.6 [+0, +4]	+12.8 [+6, +20]	-12.8 [-19, -7]
360M GRPO s0	arith+func	+99.0 [+98, +100]	-1.5 [-4, +0]	-14.5 [-20, -10]	-9.0 [-13, -5]
360M GRPO s0	linalg	+96.8 [+94, +99]	+0.0 [+0, +0]	-5.6 [-10, -2]	-8.0 [-13, -4]
360M RFT s0	arith+func	+23.0 [+17, +29]	+0.0 [+0, +0]	-3.5 [-7, +0]	-2.0 [-4, -0]
360M RFT s0	linalg	+42.4 [+34, +51]	+0.0 [+0, +0]	+9.6 [+3, +17]	-16.8 [-24, -10]
360M SFT s0	arith+func	+60.0 [+53, +67]	+1.0 [+0, +2]	-18.5 [-24, -12]	-6.5 [-10, -4]
360M SFT s0	linalg	+56.0 [+47, +65]	+3.2 [+1, +6]	-0.8 [-3, +2]	-16.8 [-23, -10]
360M SSD s0	arith+func	+15.0 [+10, +20]	+1.0 [+0, +2]	+4.5 [+0, +9]	-1.5 [-4, +0]
360M SSD s0	linalg	+43.2 [+34, +52]	+0.0 [+0, +0]	+0.8 [-5, +6]	-9.6 [-15, -5]
3B RFT s0	arith+func	+34.5 [+28, +41]	-5.5 [-8, -2]	-9.5 [-14, -4]	+0.0 [+0, +0]
3B RFT s0	linalg	+64.8 [+56, +73]	+0.0 [+0, +0]	+0.0 [-2, +2]	-5.6 [-10, -2]
3B SSD s0	arith+func	+31.5 [+25, +38]	-3.0 [-6, -1]	-3.0 [-8, +2]	-2.0 [-4, -0]
3B SSD s0	linalg	+78.4 [+71, +86]	+0.0 [+0, +0]	+0.0 [+0, +0]	-22.4 [-30, -15]

Table 6: Per-layer residual after training, $R_k = v_k(\text{free}) - v_k(S_k)$ in pp with paired-bootstrap 95% CIs (10,000 resamples, paired by prompt). Free and full-stack verify-valid rates are in Table 7 and the text. n=200 (arith+func) and n=125 (linalg).

The shape layer. Table 8 reports the unenforced layer, for the 360M SSD student. Shape diagnostics on free decoding stay at zero on arith and rise by 4.8 points on linalg, where the student emits more parse-valid programs whose operand shapes disagree. Interface mismatches rise on both pools: the base model under free decoding has none because it emits no verifier-accepted program, and after training 40 of 143 accepted arith programs and 12 of 30 linalg programs carry a signature that differs from the gold.

What each method internalizes at 360M. After SSD, free decoding produces 69.5% verifier-accepted programs on arith+func (base 0%) and exceeds the base model under the full mask (58.5%). At seed 0 RFT with the same budget is close on arith+func (free acceptance 74.5%, syntax residual 23pp against 15pp) and tied on linalg (42.4 against 43.2pp); what separates the two methods is stability across seeds, the RFT syntax residual being 31.8 ± 9.9pp against 13.5 ± 1.8 after SSD

	free		C1		C1+C2		C1+C2+C3	
model	arith	linalg	arith	linalg	arith	linalg	arith	linalg
SmolLM2-135M (base)	0/143	0/30	11/143	3/30	11/143	3/30	11/143	5/30
SmolLM2-360M (base)	0/143	0/30	20/143	2/30	22/143	4/30	26/143	4/30
SmolLM2-1.7B (base)	16/143	7/30	43/143	8/30	45/143	10/30	48/143	10/30
SmolLM3-3B (base)	9/143	6/30	46/143	7/30	49/143	15/30	55/143	16/30
Gemma-3-270M (base)	1/143	1/30	11/143	5/30	11/143	5/30	12/143	5/30
Gemma-3-1B (base)	3/143	3/30	9/143	4/30	9/143	4/30	11/143	4/30
Gemma-3-1B RFT s0	0/143	0/30	1/143	0/30	1/143	0/30	1/143	0/30
Gemma-3-1B SSD s0	4/143	5/30	4/143	6/30	4/143	6/30	4/143	6/30
SmolLM2-1.7B GRPO s0	26/143	8/30	40/143	9/30	42/143	12/30	40/143	12/30
SmolLM2-1.7B RFT s0	29/143	10/30	30/143	11/30	30/143	12/30	29/143	12/30
SmolLM2-1.7B SFT s0	56/143	5/30	56/143	6/30	56/143	7/30	58/143	7/30
SmolLM2-1.7B SSD s0	46/143	8/30	41/143	9/30	41/143	11/30	43/143	11/30
SmolLM2-135M RFT s0	16/143	3/30	17/143	3/30	17/143	3/30	17/143	3/30
SmolLM2-135M SSD s0	3/143	3/30	3/143	3/30	3/143	3/30	3/143	4/30
SmolLM2-360M GRPO s0	0/143	0/30	20/143	2/30	21/143	4/30	25/143	5/30
SmolLM2-360M RFT s0	15/143	4/30	18/143	4/30	18/143	4/30	18/143	4/30
SmolLM2-360M SFT s0	35/143	6/30	37/143	6/30	37/143	6/30	39/143	6/30
SmolLM2-360M SSD s0	21/143	5/30	19/143	5/30	19/143	5/30	20/143	5/30
SmolLM3-3B RFT s0	7/143	10/30	12/143	13/30	12/143	13/30	11/143	13/30
SmolLM3-3B SSD s0	59/143	11/30	54/143	12/30	54/143	13/30	57/143	13/30

Table 7: Functional correctness: programs matching the gold on five randomized trials, over the scorable prompts (143 arith+func, 30 linalg), one greedy decode each. The binomial interval is about 10 prompts on arith+func and about 5 on linalg, so no linalg difference here is resolved and that pair of columns reports only that nothing collapsed.

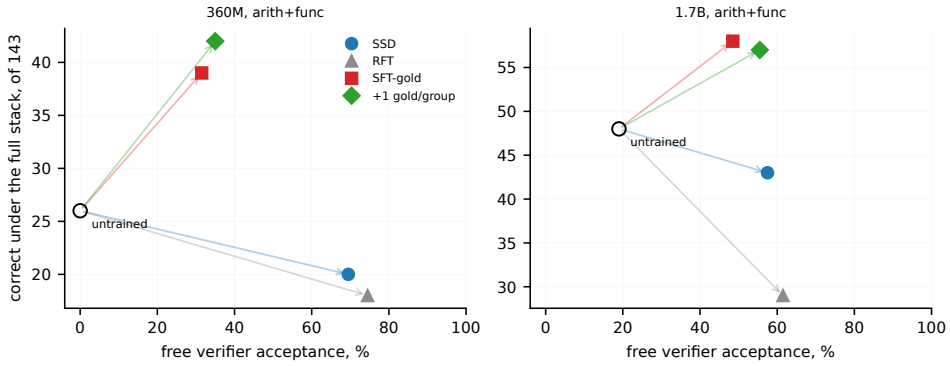


Figure 4: Free verifier acceptance against correctness under the full stack, one point per trained checkpoint on arith+func (n=143 gold-gate prompts for correctness, n=200 for acceptance). The open marker is the untrained model and each arrow runs from it to a student it was trained into. Gold-bearing supervision moves up and on-policy training moves right: the two axes separate by the source of the signal, not by the amount of training.

(Table 9 in Appendix D), and RFT correctness under the mask falls with scale (48 to 29 of 143 at 1.7B, 55 to 11 at 3B) where SSD does not. The contrast with verifier-rewarded rewriting at 3B (Fang et al., 2026) is the setting rather than the reward: that model edits IR given in the prompt in a well-represented language, ours generates from a description in a low-resource dialect. GRPO without the mask does not learn at 360M: no free rollout of the base parses. SFT-gold reaches 100% acceptance on the dev pool but 31.5% on the released arith+func pool with a syntax residual of 60pp: off-policy supervision fits the templates and does not carry to the released prompts (Figure 3; seeds in Table 9).

Seed variation. Table 9 reports every run repeated across training seeds; the main tables use seed 0 throughout. On linalg the RFT syntax residual spans 42 to 72pp across seeds against 43 to 58 after SSD.

Table 8: C4 after SSD at 360M, seed 0. v_4 is the fraction of parse-valid free-decoded programs with a shape diagnostic; the change is paired by prompt with a 10,000-resample bootstrap. Interface mismatches count verifier-accepted programs whose signature differs from the gold, under free decoding / under the full stack.

Pool	v_4 base	v_4 SSD	Δ [CI]	Mismatch base	Mismatch SSD
arith+func	0.0	0.0	+0.0 [0.0, 0.0]	0 / 30 of 143	40 / 47 of 143
linalg	0.0	4.8	+4.8 [1.6, 8.8]	0 / 12 of 30	12 / 15 of 30

run	seed	free verify %	R_1	R_2	R_3	correct free	correct full
360M RFT	0	74.5	23.0	0.0	-3.5	15.0	18.0
360M RFT	1	55.5	42.5	0.5	-8.5	17.0	18.0
360M RFT	2	67.0	30.0	0.0	-7.5	17.0	18.0
360M RFT	mean	65.7 ± 9.6	31.8 ± 9.9	0.2 ± 0.3	-6.5 ± 2.6	16.3 ± 1.2	18.0 ± 0.0
360M SFT	0	31.5	60.0	1.0	-18.5	35.0	39.0
360M SFT	1	31.0	62.0	0.5	-15.5	37.0	45.0
360M SFT	2	31.0	60.5	0.5	-20.0	36.0	44.0
360M SFT	mean	31.2 ± 0.3	60.8 ± 1.0	0.7 ± 0.3	-18.0 ± 2.3	36.0 ± 1.0	42.7 ± 3.2
360M SSD	0	69.5	15.0	1.0	4.5	21.0	20.0
360M SSD	1	71.0	11.5	1.0	4.0	22.0	20.0
360M SSD	2	70.5	14.0	1.0	4.0	22.0	20.0
360M SSD	mean	70.3 ± 0.8	13.5 ± 1.8	1.0 ± 0.0	4.2 ± 0.3	21.7 ± 0.6	20.0 ± 0.0
360M SSD + gold	0	35.0	48.5	-2.0	-7.5	36.0	42.0
360M SSD + gold	1	36.0	51.0	-1.0	-8.5	36.0	42.0
360M SSD + gold	2	35.5	51.5	-0.5	-13.0	32.0	42.0
360M SSD + gold	mean	35.5 ± 0.5	50.3 ± 1.6	-1.2 ± 0.8	-9.7 ± 2.9	34.7 ± 2.3	42.0 ± 0.0
360M SSD + coverage	0	82.0	6.0	1.0	3.5	23.0	22.0
360M SSD + coverage	1	79.0	7.0	0.0	6.5	22.0	20.0
360M SSD + coverage	2	78.5	8.5	0.0	3.5	22.0	22.0
360M SSD + coverage	mean	79.8 ± 1.9	7.2 ± 1.3	0.3 ± 0.6	4.5 ± 1.7	22.3 ± 0.6	21.3 ± 1.2
1.7B SSD	0	57.5	20.5	-2.5	-3.5	46.0	43.0
1.7B SSD	1	57.0	19.0	-3.5	0.5	51.0	55.0
1.7B SSD	mean	57.2 ± 0.4	19.8 ± 1.1	-3.0 ± 0.7	-1.5 ± 2.8	48.5 ± 3.5	49.0 ± 8.5
1.7B SSD + gold	0	55.5	33.0	-4.0	1.5	57.0	57.0
1.7B SSD + gold	1	44.0	43.0	0.5	3.0	56.0	59.0
1.7B SSD + gold	mean	49.8 ± 8.1	38.0 ± 7.1	-1.8 ± 3.2	2.2 ± 1.1	56.5 ± 0.7	58.0 ± 1.4

Table 9: Seed variation on arith+func (n=200; correctness of 143), including the two ablations repeated across seeds. One row per training seed and a mean row with the sample standard deviation over training seeds. Residuals in pp as defined in Section 3.

Ablations at 360M. Table 10 collects the seed-0 SSD student and its ablations on arith+func; the coverage pool and the gold-in-group ablations are repeated across seeds in Table 9.

Sequencing gold supervision and self-distillation. SSD started from the SFT-gold checkpoint keeps that checkpoint's released-pool correctness (37 of 143 under the full stack) while in-distribution correctness falls from 72 to 45%, and it recovers only part of the validity: free acceptance 43.5% and syntax residual 42.5 points, against 69.5% and 15.0 for SSD alone.

Second training target. Table 11 gives the LLVM IR students of Section 6. Free validity on LLVM-Spec-60 rises from 0 to 73.3% for the student trained on the lowered pool, against 46.7% transferred by the MLIR-trained student; the syntax residual is 5.0 points against 21.7 for SFT-gold, the scope residual 3.3, and correctness under the full stack is 8 of 60 for SSD against 15 for SFT-gold. The pool is the template golds lowered with `mlir-opt` and `mlir-translate` and kept when they replay under the LLVM grammar and the dominance-aware validator, 2,000 of 6,449 scanned; both students train at the MLIR budget. The MLIR-trained student transfers 46.7% free

	free	syntax residual					correct		mism.
run	verify %	R_1	vocab	struct	R_2	R_3	free	full	full
SSD (pool v1, task reward)	69.5	15.0	4.5	10.5	1.0	4.5	21	20	47
+ functional reward	75.5	10.0	3.0	7.0	0.0	6.0	21	19	54
+ coverage pool v1c	82.0	6.0	1.5	4.5	1.0	3.5	23	22	58
+ one gold per group	35.0	48.5	16.0	32.5	-2.0	-7.5	36	42	11
+ functional reward, group 16	70.0	15.0	5.5	9.5	1.0	2.0	18	19	49
+ dense scope mask	77.5	13.5	3.5	10.0	1.0	1.0	24	23	53
SFT-gold	31.5	60.0	33.5	26.5	1.0	-18.5	35	39	13
SFT-gold then SSD	43.5	42.5	23.5	19.0	1.0	-9.0	34	37	16
1.7B SSD, full fine-tuning	63.5	14.0	2.5	11.5	-1.5	2.0	52	52	16
1.7B SSD (task reward)	57.5	20.5	9.5	11.0	-2.5	-3.5	46	43	18
1.7B + functional reward	54.5	15.5	4.0	11.5	-0.5	0.0	54	53	11
1.7B + one gold per group	55.5	33.0	8.5	24.5	-4.0	1.5	57	57	6
1.7B SFT-gold	48.5	44.5	19.0	25.5	-1.5	-24.5	56	58	10

Table 10: Ablations on arith+func (n=200; correctness and mismatches of 143), seed 0. Residuals in pp as defined in Section 3; vocab and struct split R_1 into invented operations and malformed known operations.

validity to this set, against 73.3 for the student trained on it, and the scope residual after training is 3.3 points.

	valid %				residual (pp)			correct	
weights (360M)	free	C1	C1+C2	C1+C2+C3	R_1	R_2	R_3	free	full
untrained	0.0	35.0	36.7	60.0	98.3 [95, 100]	-31.7 [-43, -20]	-8.3 [-15, -2]	0	5
SSD on MLIR, transferred	46.7	55.0	55.0	60.0	20.0 [10, 30]	-10.0 [-18, -3]	13.3 [5, 22]	5	5
SSD on LLVM IR	73.3	73.3	75.0	78.3	5.0 [0, 12]	-3.3 [-10, 3]	3.3 [0, 8]	8	8
SFT-gold on LLVM IR	58.3	68.3	70.0	71.7	21.7 [12, 33]	-6.7 [-13, -2]	-3.3 [-8, 0]	14	15

Table 11: LLVM IR as a second training target on LLVM-Spec-60 (n=60). The two LLVM IR students train on the lowered template pool (2,000 programs) with the LLVM grammar, the dominance-aware scope validator, and `llvm-as` as the verifier, at the MLIR budget; the transfer row is the MLIR-trained student of Table 1. Valid = accepted by `llvm-as`; residuals as in Section 3 with 95% paired bootstrap intervals; correct = matches the reference under `lli` on five trials.

Why the scope layer resists. The class hypothesis predicted a scope residual that grows with dependency distance; the synthetic sweep does not show one (Appendix D). The data show a signal limit instead: probes read whether a name is in scope at 95% before and after training, 13 points above a probe on the token surface, patching localizes the decision to the same layers in both (Table 12 in Appendix D), and a per-token scope loss closes most of what sequence-level rejection leaves (Section 4). The in-scope decision is linearly available before and after training, which is consistent with a signal limit rather than a missing representation, in line with fine-tuning enhancing mechanisms a model already has (Prakash et al., 2024; Jain et al., 2024). Decodability alone does not establish that the model uses the signal (Sharma et al., 2026), so the per-token scope loss, which removes most of the residual, is the stronger evidence, and it is one seed at 360M.

Scope probes. Table 12 reports five-fold logistic probes on residual-stream activations of the 360M model before and after SSD, with balanced positives and a shuffled-label control, and activation patching of the use position between a clean program and a corrupted copy whose definition is renamed. A token-surface baseline runs the same probe on a hashed bag of the prefix tokens, the last four tokens, and the position; the residual probes exceed it by 13 points on the in-scope design and 18 on the name design, while the definition-differs design is solved from the surface alone and is not used as evidence. Training changes neither the best-layer accuracy nor the layer at which the patched activation recovers the clean prediction.

weights	probe	n	layer	residual % [CI]	surface %	control %
base	in scope at use	1702	27	94.7 [93.7, 95.7]	81.7	48.4
base	definition differs	774	11	98.8 [98.1, 99.5]	100.0	49.2
base	name identity	1702	31	99.6 [99.4, 99.9]	82.1	49.1
base	patching, L20/24/27/30	180		-0.01/0.68/0.94/1.00		
SSD	in scope at use	2310	27	94.8 [93.9, 95.8]	81.4	50.9
SSD	definition differs	870	19	98.6 [97.8, 99.3]	100.0	50.9
SSD	name identity	2310	31	99.2 [98.8, 99.6]	81.3	49.1
SSD	patching, L20/24/27/30	185		0.02/0.75/0.91/1.00		

Table 12: Scope probes at 360M. Five-fold logistic probes on residual-stream activations over gold and generated programs, balanced positives, with a shuffled-label control and a token-surface baseline (the same probe on a hashed bag of the prefix tokens, the last four tokens, and the position); patching reports the fraction of the clean target log-probability recovered when one layer's activation at the use position is patched from the clean run into the corrupted run. The definition-differs design is solved from the surface alone (the tokens between definition and use decide scope) and carries no evidence.

Synthetic scope sweep. A synthetic SSA language with the same four layers and a tunable dependency distance D between definition and use isolates the scope residual from MLIR specifics. After one epoch of SSD at 135M and 360M with $D \in \{2, 8, 32\}$ (Table 13), the residual conditioned on programs that parse is zero at $D = 32$ for both models and nonzero only at 135M with $D = 2$ (31 points) and 360M with $D = 8$ (5 points). The apparent growth with D over all prompts is a termination failure: at $D = 32$ free decoding falls into a repetition loop on 38 and 19% of prompts, a syntax failure the grammar's body cap prevents. The sweep does not support a residual that grows with dependency distance at these sizes, consistent with transformers learning binding chains four steps deep (Wu et al., 2025). Table 13 gives the per-cell rates behind Section D. The dev pool has 200 prompts per depth. Free outputs that fail to parse at $D = 32$ are repetition loops (a constant definition repeated until the token budget) rather than long programs: re-decoding with a 1500-token budget reproduces the same parse rates, and the parsing programs are at most 850 characters.

model	D	free parse %	v_3 free	v_3 C3	R_3 all [CI]	R_3 parse-valid (n)
135M	2	99	37.0	5.0	+32.0 [+26, +38]	+31.3 (198)
135M	8	100	0.5	0.0	+0.5 [+0, +2]	+0.0 (199)
135M	32	62	38.0	0.5	+37.5 [+31, +44]	+0.0 (124)
360M	2	100	0.0	0.0	+0.0 [+0, +0]	+0.0 (200)
360M	8	98	8.0	1.5	+6.5 [+3, +10]	+5.1 (197)
360M	32	81	19.0	0.0	+19.0 [+14, +24]	+0.0 (162)

Table 13: Synthetic scope sweep after one epoch of SSD on a synthetic SSA language with dependency distance D (n=200 dev prompts per cell). v_3 is the scope-violation rate of greedy free decoding and of the C1+C2 mask with the scope-validator retry loop; a program that does not parse counts as a violation in the all-prompt residual. The last column conditions on prompts whose free output parses.

On-policy ceiling. Table 14 gives the masked pass@k curve behind Section 7.

In-distribution correctness. Greedy correctness on the first 100 dev prompts at 360M is 21% with the task reward, 19% with the functional reward, 6% for the untrained model under the mask, and 72% for supervision on the same golds; at 1.7B the functional reward moves it from 47 to 64%.

Distance from the untrained base. Mean per-token $\log \pi_\theta - \log \pi_{\text{base}}$ over the checkpoint's own greedy decodes of 100 dev prompts, an estimate of the KL the forgetting literature relates to drift elsewhere: 0.47 nats at 360M for SSD, and at 1.7B 0.22 for the full fine-tune, 0.26 for SSD, 0.38 for SFT-gold and 0.88 for RFT. Held-out transfer is measured only for the SSD arm, so these values are reported and not related to it.

weights (360M) / k	1	4	16	64	128	256
untrained	15	25	35	40	47	49
SSD	21	23	27	–	–	–
SFT-gold	39	48	61	–	–	–
SSD + one gold per group	29	44	61	–	–	–

Table 14: Masked pass@k on arith+func (143 gold-gate prompts): prompts with at least one correct sample among k samples under the C1+C2 mask at temperature 0.8, each scored by the five-trial differential harness. The untrained model was sampled to k=256 and the students to k=16; a dash is a budget not sampled. Greedy full-stack correctness: untrained 26, SSD 20, SFT-gold 39.

Training-budget sweep. A 360M SSD student trained for eight times the step budget, with checkpoints at 126, 252, 504 and 1008 steps, gives free acceptance 75.5, 74.0, 75.0 and 74.5%, full-stack acceptance 84.0, 80.5, 80.5 and 82.5%, and full-stack correctness 19, 20, 21 and 22 of 143. These checkpoints were decoded on arith+func under the free and full stacks only, so they carry no syntax residual, and the sweep has one seed.

Sampled correctness at scale. Table 15 gives the 1.7B and 3B cells discussed in Section 7.

weights	sampled pass@1	pass@8	gap	Δ pass@1 (pp)
1.7B untrained	31.1 [23.1, 39.5]	55	23.9	–
1.7B SSD s0	40.9 [31.0, 51.5]	49	8.1	+6.8 [+1.7, +12.0]
1.7B SSD s1	44.1 [34.4, 54.5]	55	10.9	+9.1 [+4.3, +14.1]
1.7B RFT	26.2 [17.5, 35.2]	27	0.8	-3.4 [-6.7, -0.1]
3B untrained	37.1 [28.5, 46.1]	61	23.9	–
3B SSD	54.4 [43.4, 65.5]	59	4.6	+12.1 [+6.4, +17.9]
3B RFT	6.0 [2.0, 11.0]	7	1.0	-21.8 [-27.9, -15.7]

Table 15: Masked sampled correctness on arith+func (143 gold-gate prompts, 8 samples per prompt at temperature 0.8, each scored by the five-trial harness). Sampled pass@1 is the mean per-prompt fraction of correct samples, with a 95% bootstrap interval over prompts; pass@8 is the number of prompts with at least one correct sample; the gap is how many more prompts eight samples reach than one; Δ pass@1 is the paired difference from the untrained model of the same size, in percentage points, with a 95% paired bootstrap interval.

Sampled correctness. Sampled pass@1 with a bootstrap interval over prompts agrees with the greedy counts at 360M: SSD 19.5 [12.3, 27.6] and SFT-gold 38.6 [29.6, 48.1] of 143, and the dev pool has the same shape.

Support test. Table 16 counts, for every sampled base budget, the prompts each student solves that the untrained model does not reach within that budget. The untrained model was sampled to 256 completions per prompt; its curve reaches 40 prompts at k=64, 47 at 128, and 49 at 256.

Interface-given control. Table 17 decodes the released pool with the gold signature stated in the prompt. With the gold signature stated in the prompt the on-policy students stay at the untrained model's level and SFT-gold pulls further ahead, so the correctness gap is semantic rather than a signature convention.

		solved outside the untrained model's pass@k set				
student (360M)	correct	k=4	k=16	k=64	k=128	k=256
base, greedy full stack	26	5	1	1	0	0
SSD	20	2	1	0	0	0
SSD s1	20	2	1	1	0	0
SSD s2	20	2	0	0	0	0
RFT	18	2	0	0	0	0
RFT s1	18	2	0	0	0	0
RFT s2	18	2	0	0	0	0
SSD + functional reward	19	1	0	0	0	0
SSD + coverage pool	22	4	2	1	1	1
SSD + dense scope mask	23	3	1	0	0	0
SSD + one gold per group	42	24	18	17	15	15
SFT-gold then SSD	37	18	15	15	13	13
SFT-gold	39	26	21	19	16	16
SFT-gold s1	45	31	26	25	22	22
SFT-gold s2	44	29	24	22	19	19

Table 16: Support test on arith+func (143 gold-gate prompts). For each student, the prompts it solves greedily under the full stack that the untrained 360M model does not solve within k masked samples at temperature 0.8. The untrained model itself reaches k=4: 25, k=16: 35, k=64: 40, k=128: 47, k=256: 49 prompts.

	free			C1+C2+C3		
weights (360M)	accepted	correct	mismatch	accepted	correct	mismatch
untrained	0	0	0	67	28	2
SSD	62	22	5	75	24	9
RFT	62	13	33	99	27	45
SFT-gold	62	41	1	93	52	1

Table 17: Interface-given control on arith+func (150 prompts with a gold; correctness over the 143 gold-gate prompts). The gold function signature is stated in the prompt; accepted counts verifier-valid programs, mismatch counts accepted programs whose signature still differs from the gold.