CONTEXT INTO WEIGHTS: SELF-DISTILLATION FOR TEACHING A SMALL MODEL A PROGRAMMING LANGUAGE IT CANNOT WRITE

Anonymous authors
Paper under double-blind review

ABSTRACT

Small language models can transfer algorithmic knowledge across programming languages, yet may fail to express that knowledge in languages whose syntax and libraries are poorly represented in their pretraining. The standard post-training recipe, supervised fine-tuning (SFT) followed by reinforcement learning with group relative policy optimisation (GRPO), breaks at both stages here: there is no high quality corpus to fine-tune on, and when no sampled program passes its tests, GRPO has no learning signal. Yet **the missing knowledge is small and usable in context**: with a short syntax sheet in its prompt, base Qwen3-8B solves 28 of 30 elementary Ballerina programming language problems, and none without it. We therefore replace both stages with self-distillation, in which **the model with the language documentation in its context teaches the same model without it**. Self-distilled supervised fine-tuning (SD-SFT) fine-tunes the model on its own documentation-conditioned programs that pass the tests. Self-distillation policy optimisation (SDPO) then trains every token of the model's own attempts toward a teacher that also sees the documentation and the compiler's feedback. On Ballerina, where Gemma 4 E4B scores 0% pass@1, SD-SFT reaches 55.4% and SDPO **62.7%**. The same incremental ordering holds across other models and languages. Because the recipe needs no corpus in the target language, only a compiler, tests and the syntax sheet, it offers a route to adapting small models to many languages that pretraining leaves out. We release all the language agnostic training data, implementation and the model weights.

1 INTRODUCTION

Large language models can now generate complex, functional programs for long-horizon tasks in the languages that dominate public repositories, but most programming languages are not represented at comparable scale. Domain-specific languages, new general-purpose languages and languages used within a single organisation form a long tail with little public code and none of the curated instruction data that mainstream languages enjoy. Frontier models partially cover this tail by virtue of scale, yet the models that organisations can afford to fine-tune, host and keep private are typically small, and their coverage is weakest precisely in this regime. Adapting a small language model to such a programming language is therefore both a practical need and an opportunity to study how much coding competence comes from general problem-solving ability rather than language-specific knowledge.

The standard way to adapt a model to a coding task has two stages: (1) Supervised fine-tuning (SFT) on target code, (2) Reinforcement learning from executed code with a method such as group relative policy optimisation (GRPO; Shao et al., 2024; DeepSeek-AI et al., 2025). *SFT needs target-language programs.* It is hard to find a high quality corpus of target-language programs. Translating verified Python solutions needs a translator that already supports the target language (Cassano et al., 2024), and bootstrapping on the model's own verified samples (Zelikman et al., 2022; Yuan et al., 2023) needs samples that pass. *GRPO needs a passing sample.* It needs only a compiler and tests, which exist for any language (Boruch-Gruszecki et al., 2026), but its advantage is the spread of rewards within a group, which is zero when no sample passes. And even when one passes, a scalar reward says *whether* a program was right, it never specify the correct form.

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.

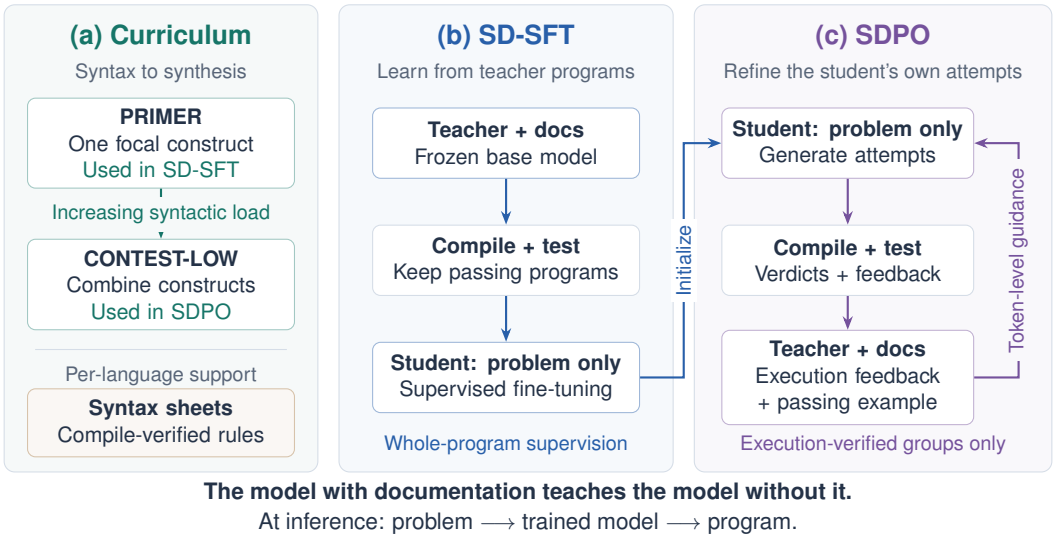


Figure 1: **From syntax to synthesis through two stages of self-distillation.** (a) Low-load PRIMER problems precede CONTEST-LOW problems that combine constructs; problem-specific syntax sheets supply the teacher's language knowledge. (b) SD-SFT trains on documentation-conditioned programs that compile and pass the tests. (c) SDPO uses execution feedback to guide the student's own attempts at the token level. Execution checks gate training in both stages. Only the teacher sees documentation, and inference uses the problem alone.

The knowledge these two stages cannot deliver is not absent, only misplaced. This distinction is illustrated by our initial experiments: with a short syntax reference in its prompt, Qwen3-8B solves 28 of 30 elementary programming problems (Table 1b), compared with none without the reference. The competence exists in the context window, and the missing piece is a way to move it into the weights. Self-distillation provides such a mechanism: the teacher is the model with access to documentation, and the student is the same model without it (Snell et al., 2022; Hübotter et al., 2026). We apply this one idea at both stages (Figure 1). In place of SFT, self-distilled supervised fine-tuning (SD-SFT) samples programs from the documentation-conditioned model, keeps only those that compile and pass the tests, and fine-tunes the model on them with the documentation removed. In place of GRPO, self-distillation policy optimisation (SDPO; Hübotter et al., 2026) trains every token of the student's own attempts toward a teacher that sees the documentation, the compiler's feedback on that attempt, and a passing program when one exists. At test time the model sees only the problem.

Both SD-SFT and SDPO can train on a problem only after a program from either the student or the teacher passes. For these models, the main obstacle is often unfamiliar syntax rather than the difficulty of solving the problem. A program compiles only if every construct is used correctly, so each unfamiliar construct creates another chance to fail even when the teacher has the documentation. We therefore order problems by *syntactic load*, the number of target-language constructs their solutions require. SD-SFT trains on elementary PRIMER problems built around one focal construct, and SDPO then trains on CONTEST-LOW problems that combine constructs. Examples within stage one are shuffled; the curriculum operates between the two stages.

The approach requires relatively little language-specific supervision. It needs no corpus in the target language and no model that already writes it: the teacher is always the model being trained. The problems and their input-output tests are language-agnostic, so the same problems and training procedure can be used for Ballerina and OCaml, with a language-specific rule inventory. The only per-language input is a short reference with one compile-checked example per construct, which the model absorbs during training and does not need at test time. Adapting a small, privately hosted model to a new language therefore requires a compact, language-specific reference rather than a target-language corpus.

Table 1: **(a) No-resource status depends on the model and language, rather than changing gradually with model size. (b) The missing competence can be activated through in-context information.** (a) Base pass@1 on Ag-LCB-X, the multilingual LiveCodeBench execution suite of Boruch-Gruszecki et al. (2026). Results for Gemma 4 E4B and Ballerina are measured in this work; all other results are taken from Boruch-Gruszecki et al. (2026). (b) Pass@1 of Qwen3-8B on the PRIMER and CONTEST-LOW problem sets, with the number of problems shown in parentheses, with and without the syntax sheet.

(a)

	Gemma 4 E4B E4B-it	Qwen3 8B	Qwen3 32B	Llama3.3 70B-Ins	DSC-v2 Lite-16B
Lua	32	11	22	25	13
Julia	33	9	26	22	12
OCaml	7	0	2	7	7
Ballerina	0	0	-	-	-

(b) Qwen3-8B on Ballerina

Prompt condition	PRIMER (30)	CONTEST-LOW (20)
Qwen3-8B		
No reference	0	0
+ Syntax sheet	28	2

We summarize our key contributions as follows:

1. **A syntax-to-synthesis curriculum.** We characterize problems by algorithmic difficulty and *syntactic load*: the number of required target-language constructs. A compile-verified rule inventory determines both load and documentation. Since contest problems raise both axes together, we introduce PRIMER for elementary, low-load training before progressing to low-rated contest problems (§2.7.1).
2. **Two-stage self-distillation from a documentation-conditioned model copy.** The recipe requires neither a target-language corpus nor a model already proficient in the language. On Ballerina, it raises Gemma 4 E4B from 0.0 to 55.4 pass@1 after stage one and 62.7 after stage two (Table 3).
3. **Benchmarks for underserved languages.** We port HumanEval to Ballerina and OCaml following MultiPL-E (Cassano et al., 2023), add Ballerina to multilingual LiveCodeBench (Boruch-Gruszecki et al., 2026), and build probes for construct correctness and avoidance (§2.7.2).
4. **Evidence favoring distillation over a GRPO hybrid.** Applying GRPO to every group containing a passing sample finishes 5.5 points below SDPO. The SDPO gain reflects reliability: compilation rises from 58.4 to 66.9, with little change in problems solved at least once (§3.2).

2 METHOD

2.1 THE SETTING: A LANGUAGE THE MODEL CANNOT WRITE

Fix a target language ℓ with a compiler and a test runner. A task is a pair $(x, \mathcal{T}_x)$ of a problem statement and hidden tests; a policy π_θ samples a completion y, a program $\mathrm{prog}(y)$ is extracted and executed, and the task is solved when every test passes. This behavioural definition (Boruch-Gruszecki et al., 2026) is all we assume about ℓ, plus a short documentation reference: no corpus, no reference implementations in ℓ. We call the setting *effectively no-resource* when the policy's initial pass rate is at or near zero, so execution feedback never fires. It is a property of the (model, language) pair.

For Qwen3-8B, performance exhibits a sharp language-dependent transition rather than a gradual dependence on model scale: Lua and Julia score nine to eleven while OCaml and Ballerina score zero (Table 1a). Ballerina has zero pass@1 for both base models used in our adaptation experiments, whereas OCaml achieves non-zero performance with Gemma 4 E4B; thus, for this model, OCaml tests the approach on a language with partial prior competence. Without the syntax sheet (panel b), a model bootstrapping on its own samples has nothing to keep, and a group-relative update has no positive contrast. With it, the model solves most elementary problems, indicating that the missing language-specific knowledge is limited and can be accessed through in-context information.

2.2 THE SHARED PRINCIPLE: THE MODEL WITH THE DOCUMENTATION TEACHES THE MODEL WITHOUT IT

Let $z(x)$ be *privileged context* for problem x: its documentation sheet $\mathrm{sheet}(x)$ (§2.3) and, in stage two, execution feedback. The *teacher* is the same network given that context, $\tilde{\pi}(\cdot \mid x) = \pi_{\tilde{\theta}}(\cdot \mid x, z(x))$. Its weights $\tilde{\theta}$ are the frozen base weights θ_0 in stage one and a moving average of the student in stage two (§2.5). The *student* is $\pi_\theta(\cdot \mid x)$ with the problem alone. Let $V_x(y) \in \{0, 1\}$ be the execution check: $\mathrm{prog}(y)$ compiles and passes every test in $\mathcal{T}_x$. Both stages move the student toward the teacher's *verified* behaviour, and they approach it from opposite sides. Stage one fine-tunes the student on the teacher's own samples that pass. Stage two scores the student's own samples, and minimises the reverse Kullback-Leibler (KL) divergence $\mathrm{KL}(\pi_\theta \,\|\, \tilde{\pi})$ at every token the student actually writes. Stage one is context distillation (Snell et al., 2022) with a compiler as the filter; stage two is SDPO (Hübotter et al., 2026) with the documentation added to the teacher's context.

2.3 THE EXTERNAL KNOWLEDGE: PER-PROBLEM DOCUMENTATION SHEETS

The teacher's advantage must be specific or it is not an advantage. With one generic sheet for every problem, the teacher solved only 7% of sampled contest problems, mostly failing on constructs the sheet did not cover (Appendix **??**). We therefore build the sheet per problem. If a reference solution exists, a rule inventory maps each construct (a `while` loop, a string split, a map, a helper function) to one compile-verified bullet giving its target-language form. A detector $\mathrm{constructs}(\cdot)$ runs over the problem's reference solution, and the sheet is the union of the bullets it requires. The bullets explain how to write a construct, but never how to solve this problem.

2.4 STAGE ONE: SELF-DISTILLED SUPERVISED FINE-TUNING (SD-SFT)

Problems arrive with a statement and tests, sometimes a reference solution in a high-resource language, never a program in ℓ. SD-SFT builds the targets with the base model π_0 itself. It samples k completions per problem with the problem's sheet in the prompt, keeps those for which $V_x(y) = 1$, and filters out completions that quote the sheet (Appendix **??**). It then *redacts* the sheet: the training instruction is the problem alone, identical to what the teacher saw minus the sheet. The loss is ordinary supervised fine-tuning on these targets,

$$\mathcal{L}_{\text{SD-SFT}}(\theta) = -\,\mathbb{E}_{x\sim\mathcal{D}_1}\,\mathbb{E}_{y\sim\pi_0(\cdot|x,\,\mathrm{sheet}(x))}\big[\,V_x(y)\,\log \pi_\theta(y \mid x)\,\big], \tag{1}$$

where $\mathcal{D}_1$ is the stage-one problem set (PRIMER, below) and $\pi_0 = \pi_{\theta_0}$ is the frozen base model, which is the stage-one teacher. Two things separate it from bootstrapping on one's own successes (Zelikman et al., 2022; Yuan et al., 2023): the sampler sees the documentation, so it produces passing programs where the unconditioned model produces none (Table 1b), and the student never does, so The model learns the difference between the two.

Choosing elementary problems with low syntactic load. Let $s(x) = |\mathrm{constructs}(x)|$ be the *syntactic load* of x, the number of inventory rules a solution requires; let p_c be the probability that the policy writes construct c in compiling form when it is required (Equation **??**) and $q(x)$ the probability that a compiling program passes the tests. A program compiles only if every construct in it is right, so if construct errors were independent,

$$\Pr[V_x = 1] \;\approx\; q(x) \prod_{c\in\mathrm{constructs}(x)} p_c \;=\; q(x)\,p^{\,s(x)} \quad (p_c \equiv p). \tag{2}$$

This heuristic motivates beginning with problems that require fewer constructs. It does not establish independence of construct errors or separate syntactic load from algorithmic difficulty. We construct PRIMER, a set of 1,179 elementary problems, each built around one focal construct (§2.7.1). Stage one trains on PRIMER; stage two draws its prompts from CONTEST-LOW, the low-rating band of Codeforces.

2.5 STAGE TWO: SELF-DISTILLATION POLICY OPTIMISATION (SDPO)

We use SDPO (Hübotter et al., 2026) with the algorithm settings of its code experiments, and change what the teacher sees. Each round takes P prompts from $\mathcal{D}_2 =$ CONTEST-LOW in a fixed

order and samples G rollouts per prompt from the current student. Each rollout is executed, giving a verdict $v_i = V_x(y_i) \in \{0, 1\}$ and a feedback string f_i: compiler diagnostics, a runtime error, or the first mismatching test. The teacher context of rollout i is $z_i = (\mathrm{sheet}(x),\ f_i,\ \mathrm{demo}(x))$: this problem's sheet, this rollout's own feedback, and a passing program from the same group when one exists (a passing rollout's own context holds the sheet alone). **Execution gate.** When no rollout passes, the teacher first attempts the problem: n teacher samples, each given the sheet and one failing rollout's feedback, are executed; if $n_{\mathrm{pass}}(x) \geq m$ of them pass, the group is distilled with weight $w(x) = n_{\mathrm{pass}}(x)/n$ and the shortest passing teacher program becomes $\mathrm{demo}(x)$; otherwise the group is dropped. When some rollout passes, $w(x) = 1$. The loss on a mini-batch B of rollouts is

$$\mathcal{L}_{\mathrm{SDPO}}(B) = \frac{1}{N_B} \sum_{i \in B} w(x_i) \sum_{t=1}^{|y_i|} \rho_{i,t}\, \kappa_{i,t}\, \mathrm{KL}_K\big(\pi_\theta(\cdot \mid x_i, y_{i,<t}) \,\big\|\, \mathrm{sg}\big[\pi_{\tilde{\theta}}(\cdot \mid x_i, z_i, y_{i,<t})\big]\big)\,, \quad (3)$$

with $N_B = \sum_{i \in B} w(x_i)\,|y_i|$. KL_K is the reverse KL over the student's top-K tokens plus one bucket for the remaining mass, and $\mathrm{sg}[\cdot]$ stops the gradient, so the teacher is a fixed target. The two per-token weights are truncated importance ratios, evaluated on the sampled token and carrying no gradient. $\rho_{i,t} = \min(\pi_{\mathrm{old}}/\pi_{\mathrm{samp}}, c)$, with π_{samp} the sampler's probability, corrects for sampling at 16-bit precision while training at 4-bit. $\kappa_{i,t} = \min(\pi_\theta/\pi_{\mathrm{old}}, c)$, with π_{old} the policy at the start of the round, is needed because a round takes several optimizer steps. The teacher $\tilde{\theta}$ is an exponential moving average of the student, $\tilde{\theta} \leftarrow (1-\eta)\tilde{\theta} + \eta\,\theta$ once per round, since a teacher that is the current student collapses the loss to a no-op (Hübotter et al., 2026). Algorithm **??** (Appendix **??**) gives the full round, and Appendix **??** the constants.

2.6 The GRPO alternatives we compare against

The obvious hybrid keeps the standard recipe wherever it applies: a group with a passing rollout carries a group-relative contrast, so give it a GRPO update and distil only the rest. We test this as the *routed* arm,

$$\mathrm{route}(x) = \begin{cases} \textsc{grpo} & \text{if some rollout passes,} \\ \textsc{sdpo} & \text{if none passes and } n_{\mathrm{pass}}(x) \geq m, \\ \textsc{drop} & \text{otherwise,} \end{cases} \quad (4)$$

where a GRPO group takes the GRPO baseline objective of Hübotter et al. (2026): the clipped surrogate of proximal policy optimisation (PPO; Schulman et al., 2017) against π_{old}, weighted by the same $\rho_{i,t}$, with advantages $A_i = v_i - \bar{v}$ ($\bar{v}$ the group's mean verdict) and no KL penalty. Everything else equals SDPO. *GRPO only* drops every group without a pass. Both arms start from the same SD-SFT checkpoint as SDPO, on the same prompts in the same order.

2.7 Problem sets and benchmarks

2.7.1 Training problem sets

Ballerina uses three problem sets, profiled in Table 2 on the two axes of §2.4: difficulty and syntactic load. PRIMER is the authored low-load set motivated there; it ships a statement and tests but no reference solution in any language. CONTEST-LOW and CONTEST-MIXED are Codeforces problems, the first from the low-rating band and the second spanning ratings 800 to 3,400. They ship a Python reference and input/output tests, and no target-language program. Stage one trains on PRIMER and stage two draws its prompts from CONTEST-LOW; CONTEST-MIXED, on which no stage trains, shows how load grows with rating. OCaml uses the same construction with its own rule inventory (Appendix **??**).

2.7.2 Evaluation benchmarks

The primary benchmark is HumanEval (Chen et al., 2021), ported to each target language as in MultiPL-E (Cassano et al., 2023): 159 Ballerina, 158 OCaml and 147 Fortran function-completion problems. The harness compiles the extracted program together with the problem's tests and calls the named function directly. It shares no source with the training sets, which are authored or taken from Codeforces. For the motivating measurement and the comparison with the published baseline

Table 2: **The three problem sets on the two axes** (Ballerina). *Rating*: Codeforces rating, median and share at 1800 or above; PRIMER is authored and has none, CONTEST-LOW is rated only on the 75 problems it shares with CONTEST-MIXED (*, none above 1400). *Load*: $s(x)$ over the 33-rule inventory, mean (median), from the Python reference for the Codeforces sets and from the union of verified programs for PRIMER (463 problems have one), an upper bound. *Covered*: rules required by at least five problems.

				Difficulty		**Syntactic load**		
Set	**Used in**	**Source**	**Problems**	**Rating**	**≥1800**	$s(x)$	$s \geq 5$	**Covered**
PRIMER	stage one	authored	1,179	-	-	4.4 (4)	47%	22/33
CONTEST-LOW	stage two	Codeforces, low band	490	1100*	0%*	5.4 (5)	64%	30/33
CONTEST-MIXED	-	Codeforces, 800-3400	498	1600	36%	7.1 (7)	79%	30/33

(Tables 1a and 4) we also score Ag-LCB-X, the multi-language LiveCodeBench execution suite of Boruch-Gruszecki et al. (2026). It judges a whole program by its input/output behaviour and resists contamination (Jain et al., 2025). Because the two benchmarks call a program differently, we never average them (Appendix **??**). Two further evaluations are controls rather than targets. The original Python HumanEval checks that learning a new language costs no general coding skill (§2.1). A set of 90 authored probes, disjoint from every training set, measures per construct of the inventory whether the model writes it in compiling form, which pass@1 cannot separate from avoiding the construct (Appendix **??**).

3 EXPERIMENTS

Setup. We adapt Gemma 4 E4B to Ballerina, on which it scores zero, and to OCaml, which it partly knows. Qwen3-8B (Yang et al., 2025) is a second base model, and the one on which the only published baseline for these languages is reported (Boruch-Gruszecki et al., 2026). Both stages train a low-rank adapter (LoRA; Hu et al., 2022) on a 4-bit base model (Dettmers et al., 2023) on one GPU. We evaluate on the benchmarks of §2.7.2, mainly HumanEval ported to each language, and report pass@1 and the compile rate. Stage two compares SDPO with the two GRPO variants of §2.6, all started from the same SD-SFT checkpoint; untrained baselines give the base model the full language reference or retrieved documentation (Lewis et al., 2020). Intervals are paired per-problem bootstrap 95% intervals; Appendices **??** and **??** give every setting.

3.1 STAGE ONE TEACHES THE LANGUAGE, THROUGH COMPILATION

On Ballerina the base model scores 0.0 zero shot and 1.3 with the complete language reference in its prompt; SD-SFT reaches 55.4 with no reference at all (Table 3). On OCaml, an earlier matched comparison, in which both the base model and a separate SD-SFT checkpoint receive the same fixed syntax section, shows a gain of 13.1 points (95% CI $[+7.8, +18.4]$; Appendix **??**). The gain is almost all compile rate: among programs that compile, the pass rate barely moves ($90.8 \rightarrow 93.4$). The documentation is needed at generation time. Without the sheet the base model writes no passing program (Table 1b), so bootstrapping on its own samples (Zelikman et al., 2022) would train on nothing. The Ballerina SD-SFT checkpoint still fails primarily at the language surface, with compile errors causing nearly all failures. (Appendix **??**).

3.2 STAGE TWO IMPROVES BALLERINA EXECUTION RELIABILITY

SDPO improves reliability over stage one, while routing solved groups to GRPO yields smaller gains. On Ballerina, SDPO gains 7.3 points over SD-SFT (95% CI $[+3.5, +11.0]$); the routed arm, differing only in Equation 4, gains 1.8 ($[-1.7, +5.3]$) and finishes 5.5 points lower ($[-9.6, -1.3]$). OCaml shows the same ordering: SDPO gains 4.9 ($[-0.2, +9.9]$), and routing trails at every scored checkpoint (Appendix **??**). Routing sends nearly nine in ten trained rollouts to GRPO; its adapter moves roughly half as far and produces fewer passing rollouts on the same training prompts. The SDPO gain primarily reflects reliability: compilation rises from 58.4 to 66.9, while both the pass

Table 3: **Main result.** pass@1 and compile rate (Comp.), in percent, on HumanEval ported to each language. The frontier model rows are zero-shot context. Every other row is Gemma 4 E4B: the base-model rows vary only the prompt.

	Ballerina		OCaml	
Arm	**pass@1**	**Comp.**	**pass@1**	**Comp.**
Frontier models, zero shot				
`claude-haiku-4-5`	0.4	0.8	97.7	98.5
`gpt-5.6-luna`	22.0	22.0	93.7	98.3
Base model, no training				
Zero shot	0.0	0.0	43.7	46.8
+ full reference	1.3	1.3	32.3	35.4
+ retrieval (RAG)	0.0	0.0	47.5	52.5
Stage one				
SD-SFT without curriculum	8.8	8.8	42.4	44.3
SD-SFT with curriculum	55.4	58.4	51.9	54.9
Stage two, from SD-SFT				
+ routed: GRPO where a rollout passes	57.2	60.2	55.1	59.3
+ SDPO	**62.7**	**66.9**	**56.8**	**61.4**

Table 4: **Qwen3-8B against the published baseline.** pass@1 in percent on ported HumanEval (HE) and Ag-LCB-X (LCB). The Agnostics row (Boruch-Gruszecki et al., 2026), trained with group-relative RL on their Ag-Codeforces-X problems, and the base model's OCaml and Fortran LCB cells are the numbers that paper reports. -: no published number. Our rows use a different sampler setting from Table 3 (Appendix **??**), so compare within this table.

	OCaml		Fortran	
Arm (Qwen3-8B)	**HE**	**LCB**	**HE**	**LCB**
Base model, zero shot	18	0	42	0
Agnostics RL on Ag-Codeforces-X	-	7	-	17
SD-SFT	56	**23**	56	18
+ SDPO	**61**	22	**59**	**19**

rate among compiling programs and the number of problems solved within three samples remain largely unchanged.

3.3 A SECOND BASE MODEL, AND THE PUBLISHED BASELINE

The only published result for these languages is Agnostics (Boruch-Gruszecki et al., 2026), which trains Qwen3-8B with group-relative RL and scores it on Ag-LCB-X (Table 1a). Table 4 scores our two stages on the same base model and suite. On Ballerina, where base Qwen3-8B compiles no program, SD-SFT reaches 45.5 pass@1. As on Gemma 4 E4B, pass@1 is about 0.94 times the compile rate, so the gap between the two models is compilation.

Table 4 extends the evaluation to Qwen3-8B. On HumanEval, SD-SFT raises OCaml pass@1 from 18 to 56 and Fortran from 42 to 56; SDPO further raises these scores to 61 and 59. The second-stage changes on Ag-LCB-X: OCaml moves from 23 to 22 and Fortran from 18 to 19. Thus, the adaptation gains extend across models and languages, while the incremental benefit of online distillation depends on the benchmark. On Ballerina, Qwen3-8B SD-SFT reaches 45.5 pass@1 at a 48.4% compile rate. Also this covers both Dense and MoE SLM architectures.

Table 5: **Which errors each stage removes.** Ballerina, every sample classified by its first compiler error, per 100 samples (columns sum to 100). No sample timed out or lacked code.

	Base	**SD-SFT**	**+SDPO**
Nonexistent function, module or type	60.4	8.6	6.1
Grammar borrowed from other languages	34.6	12.6	8.4
Reserved word used as a name	0.0	3.6	2.7
Optional values and error handling	0.6	4.7	4.2
Strict typing, no implicit conversion	1.9	4.9	6.3
Immutability and assignment rules	1.9	5.2	4.0
Other compile error	0.6	1.9	1.5
Wrong answer (compiles, fails a test)	0.0	3.0	4.2
Pass	0.0	55.4	62.7

4 Discussion

Why distil rather than prompt. Given the complete Ballerina reference in its prompt, each frontier model gains at least 30 points of pass@1; Gemma 4 E4B gains 1.3, which is not significant, and retrieved passages add nothing (Appendix **??**). The small model copies the reference's surface without the rule behind it: with the reference, half of its programs import a core library module that must be quoted because it is named after a type keyword, and most of those imports fail to compile. Instead, we provide a short, compile-verified form for each construct the problem needs in a per-problem sheet (Table 1b). Selecting a sheet's bullets needs a reference solution, so the sheet cannot be served at test time (§2.3); distillation moves its content into the weights during training, when reference solutions exist. For scale only, since the two sides were sampled differently, SDPO with no reference scores above both frontier models given the full one.

What the training changes. The measured accuracy improvements are accompanied primarily by higher compilation rates. Stage two raises Ballerina's compile rate by 8.5 points with little change in the pass rate among compiling programs; stage one shows the same pattern on OCaml (§3.1; Appendix **??**). The main errors are nonexistent names and grammar borrowed from other languages (Table 5). Stage one removes most of both; stage two reduces them by another 2.5 and 4.3 errors per 100 samples, respectively. The routed arm reduces borrowed grammar but shows little improvement on nonexistent names (Appendix **??**). Compiler feedback and the reference sheet may help correct these names, although we do not isolate their contributions. Nearly 9/10 of the remaining failures are compile errors.

Where the teacher's signal lands. GRPO assigns a program-level advantage, whereas distillation provides token-level supervision. We expected teacher-student divergence to concentrate on the construct rejected by the compiler. Instead, the top 5% of tokens carry 80% of the divergence, concentrated in reasoning; the flagged line receives only 0.61 times its proportional share of program divergence (95% CI $[0.17, 0.85]$; Figure 2; Appendix **??**). The teacher receives compiler messages without line numbers, which may limit its ability to locate errors. Including these locations is a concrete, untested extension.

5 Related Work

Prior work answers an under-represented language by building its data, translating verified Python solutions into it (Cassano et al., 2024; Giagnorio et al., 2026), which presupposes a translator that already writes it; or by execution rewards (Shao et al., 2024; DeepSeek-AI et al., 2025; Yu et al., 2025), made language-agnostic by Agnostics (Boruch-Gruszecki et al., 2026), which presuppose a passing sample and give a language reference no route into the update. Bootstrapping on one's own successes (Zelikman et al., 2022; Yuan et al., 2023) inherits the same precondition. Our stage one is closest to context distillation (Snell et al., 2022), which fine-tunes a model to produce without a prompt what it produces, and to STaR's rationalisation (Zelikman et al., 2022), which generates with the answer as a hint and trains without it; SD-SFT makes the same move with documentation as the context and a compiler as the filter. Self-distillation has also kept fine-tuning targets in the model's



(a) the whole rollout: mean divergence per stretch of tokens; red ticks mark the compiler-flagged line

```
   ...
43     int N = 0;
44     int D = 0;
45     string tok = "";
46     foreach string:Char c in line1 {
47         if c == " " || c == "\r" || c == "\n" {
48             if tok.length() > 0 {
49                 int val = check int:fromString(tok);
50 ▶               if (N == 0) N = val;
51                 else D = val;
52                 tok = "";
53             }
54         } else {
55             tok = tok + c;
56         }
57     }
   ...
```

Compiler's first error (line 50, marked ▶)

```
missing open brace token
```

Teacher's context: the problem's sheet and this rollout's compiler feedback.

Shade: per-token divergence between student and teacher, from white (none) to largest in the panel.

(b) the program around the flagged line, each token shaded

Figure 2: **Where the teacher's signal lands.** The median rollout of our measurement: a student program that fails to compile on a stage-two training prompt, at the stage-two starting weights.

own distribution (Yang et al., 2024), and served as an on-policy teacher conditioned on demonstrations for continual learning (Shenfeld et al., 2026). Our stage two is SDPO (Hübotter et al., 2026), which builds on privileged information (Vapnik & Vashist, 2009) and distillation (Hinton et al., 2015); we add the documentation and an execution gate. Our contribution is a documentation-conditioned adaptation recipe using these established objectives, together with evidence that SDPO outperforms the tested routing variant on Ballerina at the reported endpoint. In-context use of a reference that contradicts the prior grows with scale (Wei et al., 2023), predicting that in-context learning (Brown et al., 2020) and retrieval (Lewis et al., 2020) fail at ours. Curriculum learning orders by difficulty (Bengio et al., 2009); we begin with elementary problems before progressing to contest problems in the online stage. Appendix **??** treats every line in full.

6 Limitations

Extending the provided training data to additional language constructs requires a human-written, compile-verified reference entry for each new construct. Our training covers general-purpose constructs used in competitive programming problems; it does not cover domain-specific constructs or external libraries. We evaluate only single-file, function-level problems and do not establish the method's effectiveness on repository-level tasks.

7 Conclusion

We show that a small model can learn to generate Ballerina programs without a pre-existing Ballerina training corpus. SD-SFT first trains on programs generated with a short syntax reference and verified by compilation and tests. SDPO then refines the model using token-level supervision from a copy of itself given the reference and compiler feedback. The trained student uses no documentation at inference. On Ballerina, these stages raise pass@1 from 0.0% to 55.4% and then 62.7%. Routing groups with passing samples to GRPO instead finishes at 57.2%. The second-stage improvement mainly comes from producing compilable programs more consistently. OCaml shows the same ordering. These results support a concrete approach to language adaptation: use a compiler, tests, and short, verified examples of language constructs to generate training supervision, then distill that supervision into the model.

AI USE STATEMENT

A large language model assisted with this work in two roles, both under author supervision. (1) *Writing.* It was used to check grammar and to improve the clarity and tone of the text, and to draft and revise passages from the authors' notes and results. The authors reviewed and edited every sentence and checked every claim against the underlying logs. (2) *Engineering.* It was used to identify bugs in the implementation, to help build the problem-specific documentation sheets, and to contribute to parts of the training, evaluation and analysis code; the authors reviewed all of it. The model was not used to generate, impute or extrapolate any experimental number: every value in this paper was produced by executing the released code. The scientific claims, the experimental design and the interpretation of the results are the authors' own, and the authors take full responsibility for the content of the paper.

ETHICS STATEMENT

This work fine-tunes openly licensed models on problems drawn from public programming-contest archives and on synthetic problems authored for the project; it involves no human subjects and no personal data.

REPRODUCIBILITY STATEMENT

The method is specified in Section 2, with the two stage losses as Equations 1 and 3, one stage-two round as Algorithm **??**, and the GRPO variants we compare against as Equation 4. Appendix **??** lists every hyperparameter of the reported runs. Appendix **??** documents how each corpus was built, how the documentation sheets are derived and compile-verified, and how sheet leakage into the training targets is filtered. Appendix **??** gives the evaluation protocol (sampler, prompt, extraction rule, execution harness) and how much each of these moves results on the same weights. Code, run configurations, datasets, documentation sheets, probe sets, benchmark ports for the three languages, and all adapter checkpoints are released.

REFERENCES

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th International Conference on Machine Learning*, 2009.

Aleksander Boruch-Gruszecki, Yangtian Zi, Zixuan Wu, Tejas Oberoi, Carolyn Anderson, Joydeep Biswas, and Arjun Guha. Agnostics: Learning to synthesize code in any programming language with a universal reinforcement learning environment. In *International Conference on Learning Representations*, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/819977c0a95458911bbfd9e5b5115018-Abstract-Conference.html.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q. Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. MultiPL-E: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 49(7):3675–3691, 2023. doi: 10.1109/TSE.2023.3267446.

Federico Cassano, John Gouwar, Francesca Lucchetti, Claire Schlesinger, Anders Freeman, Carolyn Jane Anderson, Molly Q. Feldman, Michael Greenberg, Abhinav Jangda, and Arjun Guha. Knowledge transfer from high-resource to low-resource programming languages for code LLMs. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA2):677–708, 2024. doi: 10.1145/3689735.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *Nature*, 645:633–638, 2025. doi: 10.1038/s41586-025-09422-z.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, pp. 10088–10115, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1feb87871436031bdc0f2beaa62a049b-Abstract-Conference.html.

Alessandro Giagnorio, Alberto Martin-Lopez, and Gabriele Bavota. No resource, no benchmarks, no problem? Evaluating and improving LLMs for code generation in no-resource languages. *IEEE Transactions on Software Engineering*, 52(9):2774–2788, 2026. doi: 10.1109/TSE.2026.3703553.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. URL https://arxiv.org/abs/1503.02531.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Jonas Hübotter, Frederike Lübeck, Lejs Behric, Anton Baumann, Marco Bagatella, Daniel Marta, Ido Hakimi, Idan Shenfeld, Thomas Kleine Buening, Carlos Guestrin, and Andreas Krause. Reinforcement learning via self-distillation. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=QkfkxyRizZ.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/94074dd5a072d28ff75a76dabed43767-Abstract-Conference.html.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Idan Shenfeld, Mehul Damani, Jonas Hübotter, and Pulkit Agrawal. Self-distillation enables continual learning. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=qA6FgH0nnZ.

Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv preprint arXiv:2209.15189*, 2022. URL https://arxiv.org/abs/2209.15189.

Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, 22(5–6):544–557, 2009.

Jerry Wei, Jason Wei, Yi Tay, Dustin Tran, Albert Webson, Yifeng Lu, Xinyun Chen, Hanxiao Liu, Da Huang, Denny Zhou, and Tengyu Ma. Larger language models do in-context learning differently. *arXiv preprint arXiv:2303.03846*, 2023. URL https://arxiv.org/abs/2303.03846.

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

Zhaorui Yang, Tianyu Pang, Haozhe Feng, Han Wang, Wei Chen, Minfeng Zhu, and Qian Liu. Self-distillation bridges distribution gap in language model fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1028–1043, 2024. doi: 10.18653/v1/2024.acl-long.58.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, et al. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, volume 38, pp. 113222–113244, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/a4277440d50f1f15d2cb4c14f7e0c0d2-Abstract-Conference.html.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023. URL https://arxiv.org/abs/2308.01825.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.