

1 .1 Prompt Template

2 To keep supervision minimal while still guaranteeing a well-formed trajectory, we supply every query through a fixed prompt template. The template makes three regions explicit. First, a `<think>...</think>` block reserves space for free-form chain-of-thought tokens; earlier experiments showed that the length of this block grows in concert with accuracy and code use, echoing the positive correlations reported in the Agent RL Scaling Law. Second, any text enclosed in triple back-ticks is interpreted as Python, executed in the sandbox, and its stdout is appended to the dialogue, allowing the policy to incorporate deterministic computations into its reasoning. Third, the final prediction is wrapped in `<answer>...</answer>` and repeated inside `\boxed{}` so the evaluator can extract a single token span and assign the binary reward defined in the previous subsection.

12 Figure 1 gives a concrete example. Apart from replacing the placeholder `{query}` with the current task, the template is left unchanged for all models, datasets, and training steps, ensuring that performance gains arise from policy improvement rather than prompt tuning.

```
A conversation between User and Assistant. The User asks
a question, and the Assistant solves it. The Assistant
first thinks about the reasoning process in the mind and
then provides the User with the answer. The reasoning
process is enclosed within <think> </think> and answer
is enclosed within <answer> </answer> tags, respectively,
i.e., <think> reasoning process here </think> <answer>
answer here </answer>.
User: You can use Python code during the solution
process, and the code will be executed immediately
and the result will be returned. You must put your
answer inside <answer> </answer> tags, i.e., <answer>
answer here </answer>. And your final answer will be
extracted automatically by the \boxed{} tag. \nThis is the
problem:{query}\nAssistant: <think>
```

Figure 1: The prompt template used to guide the LLM’s generation process. `{query}` is replaced with the specific mathematical problem. The template encourages structured reasoning and answer extraction, while explicitly permitting Python code usage.

15 .2 Reward Function

16 Following the ZeroRL philosophy, the training signal is restricted to a single binary outcome that depends only on the final answer. This choice mirrors the empirical finding reported earlier: increases in accuracy, code-execution frequency, and response length arise jointly when the agent is judged solely on end-task success. For an input problem x and a generated trajectory y , let $a_{\text{pred}}(y)$ denote the expression extracted from the `\boxed{}` tag and let $a_{\text{gold}}(x)$ be the ground-truth solution. The reward is

$$R(x, y) = \begin{cases} +1, & \text{if } \text{IsCorrect}(a_{\text{pred}}(y), a_{\text{gold}}(x)), \\ -1, & \text{otherwise.} \end{cases}$$

22 The function `IsCorrect` first normalises both strings and, for numerical outputs, verifies exact numerical equality; for symbolic answers it defaults to case-sensitive string matching. No auxiliary reward is granted for intermediate reasoning tokens or for successful execution of Python snippets. Consequently, the policy must discover on its own that invoking code can increase the likelihood of producing the correct boxed answer—a behaviour that, as demonstrated in the experiments, scales predictably with additional training steps and larger interaction budgets.

28 .3 Generalisation to Larger Inference Budgets

29 Table 1 and Figure 2 explore what happens when the interaction ceiling is fixed during training but lifted at test time. The first block of results comes from the DeepMath dataset where the policy

31 was trained with an upper limit of four tool calls. When the evaluation limit is raised from zero
32 to four the average climbs by more than fourteen percent. A further rise to twelve calls pushes the
33 mean to thirty two percent yet the gain between twelve and sixteen calls is negligible. Contest style
34 datasets such as AIME respond most strongly, moving from sixteen percent on the zero call setting
35 to forty percent when eight or twelve calls are allowed. Math500 increases by only eleven percent
36 across the same range, evidence that symbolic proof tasks saturate earlier than numeric contest
37 questions. Throughout this sweep the code ratio grows from zero to just under three, echoing the
38 earlier conclusion that the agent prefers concise high-value snippets and rarely needs to execute
39 more than two blocks.

40 The second block uses the more heterogeneous orz-57k dataset with a training cap of two. A single
41 extra call at evaluation almost doubles the average, a pattern that continues up to eight calls where
42 the curve flattens near thirty three percent. The improvement then oscillates, suggesting that beyond
43 a modest allowance additional opportunities are rarely exploited. Code ratio rises quickly to four and
44 then levels off which supports the same interpretation: most problems are solved with one decisive
45 execution, and extra capacity mainly benefits a small tail of harder instances.

46 Taken together these results confirm that policies trained under a tight budget can transfer smoothly
47 to inference regimes that grant more freedom. Accuracy improves rapidly up to roughly eight to
48 twelve calls and then plateaus, a shape that mirrors the positive but saturating relationship between
49 interaction count and performance described by the Agent RL Scaling Law.

50 **.4 Consolidated Insights**

51 The analysis across both main text and appendix paints a consistent picture of how ZeroTIR acquires
52 and exploits tool use. The fixed prompt template supplies just enough structure for the agent to
53 plan privately, call the Python executor when useful, and surface a clean boxed answer. The reward
54 signal, a single binary flag tied only to that boxed answer, funnels learning pressure toward outcomes
55 rather than procedures. Under this sparse guidance the training curves reveal three linked effects.
56 First, as soon as the agent begins to solve tasks it lengthens its internal reasoning trace and inserts
57 short code snippets more frequently, exactly the co-movement predicted by the scaling law. Second,
58 raising the interaction cap during training yields monotonic accuracy gains up to about twenty calls,
59 after which the curve bends toward saturation. Third, a model trained under a tight cap generalises
60 smoothly when given extra calls at test time, realising most of the remaining headroom within eight
61 to twelve executions while keeping the average number of calls per answer close to two. Together
62 these results show that spontaneous tool invocation emerges as a stable, sample-efficient strategy
63 and that its benefits scale predictably with both model capacity and allowed interaction budget.

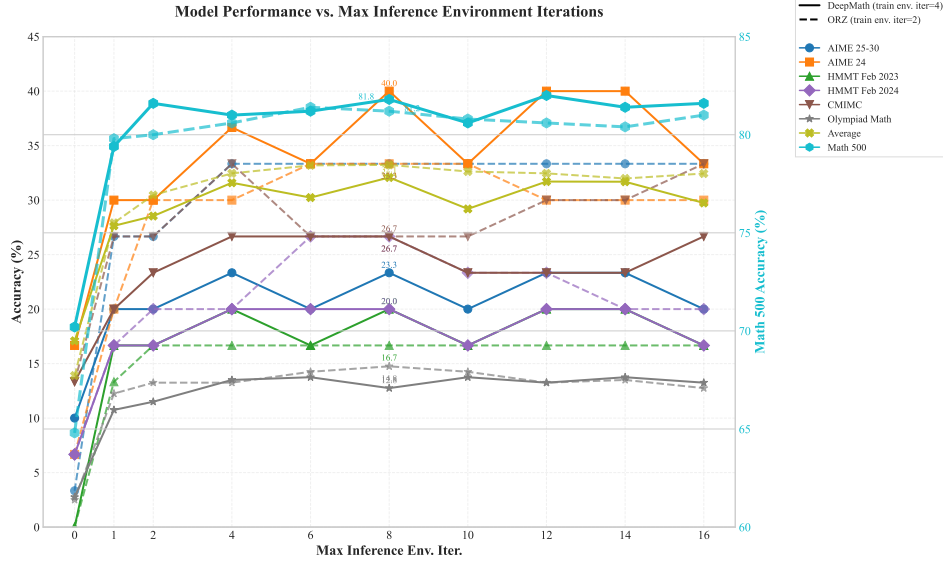


Figure 2: Average accuracy as a function of the maximum number of tool calls allowed during evaluation. Solid curve shows DeepMath with a training cap of four, dashed curve shows Orz-57k with a training cap of two.

Pred-it	AIME		HMMT Feb.		Others			Avg	Code ratio
	25	24	25	24	CMIMC	Olymp	MATH500		
Reinforce++ 7B (DeepMath, step=400, train-it=4)									
0	10.00%	16.67%	0.00%	6.67%	13.33%	2.75%	70.2%	17.09%	0.00
1	20.00%	30.00%	16.67%	16.67%	20.00%	10.75%	79.4%	27.64%	2.35
2	20.00%	30.00%	16.67%	16.67%	23.33%	11.50%	81.6%	28.54%	2.40
4	23.33%	36.67%	20.00%	20.00%	26.67%	13.50%	81.0%	31.60%	2.59
6	20.00%	33.33%	16.67%	20.00%	26.67%	13.75%	81.2%	30.23%	2.58
8	23.33%	40.00%	20.00%	20.00%	26.67%	12.75%	81.8%	32.08%	2.59
10	20.00%	33.33%	16.67%	16.67%	23.33%	13.75%	80.6%	29.19%	2.59
12	23.33%	40.00%	20.00%	20.00%	23.33%	13.25%	82.0%	31.70%	2.67
14	23.33%	40.00%	20.00%	20.00%	23.33%	13.75%	81.4%	31.69%	2.73
16	20.00%	33.33%	16.67%	16.67%	26.67%	13.25%	81.6%	29.74%	2.75
Reinforce++ 7B (Orz-57k, step=200, train-it=2)									
0	3.33%	6.67%	0.00%	6.67%	13.33%	2.50%	64.8%	13.90%	0.00
1	26.67%	20.00%	13.33%	16.67%	26.67%	12.25%	79.8%	27.92%	4.10
2	26.67%	30.00%	16.67%	20.00%	26.67%	13.25%	80.0%	30.46%	4.34
4	33.33%	30.00%	16.67%	20.00%	33.33%	13.25%	80.6%	32.45%	3.08
6	33.33%	33.33%	16.67%	26.67%	26.67%	14.25%	81.4%	33.19%	3.37
8	33.33%	33.33%	16.67%	26.67%	26.67%	14.75%	81.2%	33.23%	3.68
10	33.33%	33.33%	16.67%	23.33%	26.67%	14.25%	80.8%	32.63%	4.37
12	33.33%	30.00%	16.67%	23.33%	30.00%	13.25%	80.6%	32.45%	3.86
14	33.33%	30.00%	16.67%	20.00%	30.00%	13.50%	80.4%	31.99%	4.14
16	33.33%	30.00%	16.67%	20.00%	33.33%	12.75%	81.0%	32.44%	4.48

Table 1: Performance experiment of fixing the upper limit of interaction in training environment and scaling the upper limit of interaction in inference environment.