Training Small Language Models for Code-Based Information Search

Aaron Tamte
tamte.aa@northeastern.edu

GitHub Repository: https://github.com/2025-F-CS6220/project-slm-rl-search

1 Project Overview

We train a 4-billion parameter model to perform information search by writing Python code. Given a question, the model generates code that queries a Wikipedia corpus, navigates to relevant sections, and extracts answers—achieving 42.68% accuracy on held-out questions, a 3.7× improvement over the untrained baseline.

Unlike retrieval-augmented generation (RAG), which typically issues a single query, or tool-use frameworks that constrain models to one function call per turn, code generation lets the model compose loops, conditionals, and string operations into multi-step search strategies. Training proceeds in two stages: supervised fine-tuning (SFT) on expert trajectories teaches basic search patterns, then reinforcement learning (RL) optimizes for answer correctness. SFT alone reaches 28.13%; RL adds 14.5 percentage points.

The model learns genuine search behaviors: formulating targeted queries, iterating through results with programmatic filters, and adapting based on intermediate outputs.

2 Input Data

This work uses two datasets from HuggingFace: a Wikipedia corpus for retrieval and a question-answer dataset for training and evaluation.

2.1 Search Corpus

The corpus contains 2,590 Wikipedia pages from the rare-wiki-pages dataset.[1] Pages cover niche topics unlikely to be memorized during pretraining. Pages are markdown-formatted with hierarchical section headers.

Example corpus entry:

```
{
  "id": "caroline_polachek",
  "title": "Caroline Polachek",
  "content": "# Caroline Polachek\n\n**Caroline Polachek** (born 1985)..."
}
```

[1] https://huggingface.co/datasets/willcb/rare-wiki-pages

2.2 Question-Answer Pairs

The dataset contains 478 factoid question-answer pairs from wiki-trivia-questions-v4.[2]

```
{
  "question": "Which university did Miles Teller attend for his BFA?",
  "answer": "New York University",
  "filename": "Miles Teller.md"
}
```

2.3 Data Split

We use the first 80% (382 questions) for training and the remaining 20% (96 questions) for evaluation.

3 Problem

We train a small language model to answer factoid questions by writing Python code that searches a Wikipedia corpus. The model has no direct access to article content; it must retrieve information through code execution. This setup prevents the model from relying on memorized facts and forces it to learn genuine search behavior.

The model writes Python code that executes in a sandboxed environment with access to three functions:

- `search_pages(query)` — Returns Wikipedia pages matching the query string using semantic search.
- `view_sections(page_id)` — Returns the hierarchical section structure of a given page.
- `read_section(section_id)` — Returns the full text content of a specific section.

After execution, the sandbox returns stdout to the model, which then either generates more code or produces a final answer. This iterative loop continues until the model outputs an answer or reaches the maximum number of turns. Training proceeds in two stages: supervised fine-tuning on expert search trajectories teaches the model basic search patterns, then reinforcement learning optimizes for answer correctness. Unlike fixed tool-use formats that allow one function call per turn, code generation lets the model compose loops, conditionals, and string operations in a single generation.

4 Evidence of Success

4.1 Results

Our trained model achieves 42.68% accuracy on held-out questions, compared to 11.46% for the untrained Qwen3-4B baseline. SFT alone achieves 28.13%; RL adds 14.5 percentage points. We include Qwen3-1.7B to show scaling effects and GPT-5 as an upper bound.

We use accuracy as the primary metric since each question has a single correct answer.

[2] `https://huggingface.co/datasets/willcb/wiki-trivia-questions-v4`

Model	Parameters	Accuracy
Qwen3-1.7B (untrained)	1.7B	5.21%
Qwen3-4B (untrained)	4B	11.46%
Qwen3-4B (SFT)	4B	28.13%
Qwen3-4B (SFT + RL)	4B	42.68%
GPT-5 (reference)	∼1T	82.72%

Table 1: Model performance comparison on held-out evaluation set.

4.2 Method

Supervised Fine-Tuning (SFT). We train on expert trajectories, successful search episodes collected by running GPT-5 on training questions. The objective is standard next-token prediction (cross-entropy loss), computed only on assistant responses:

$$\mathcal{L}_{\text{SFT}} = -\sum_{t \in \text{assistant}} \log p_\theta(x_t | x_{<t}) \tag{1}$$

We use 81 deduplicated trajectories, training for 8 epochs with LoRA ($r = 16$, $\alpha = 32$) and learning rate 1×10^{-5}.

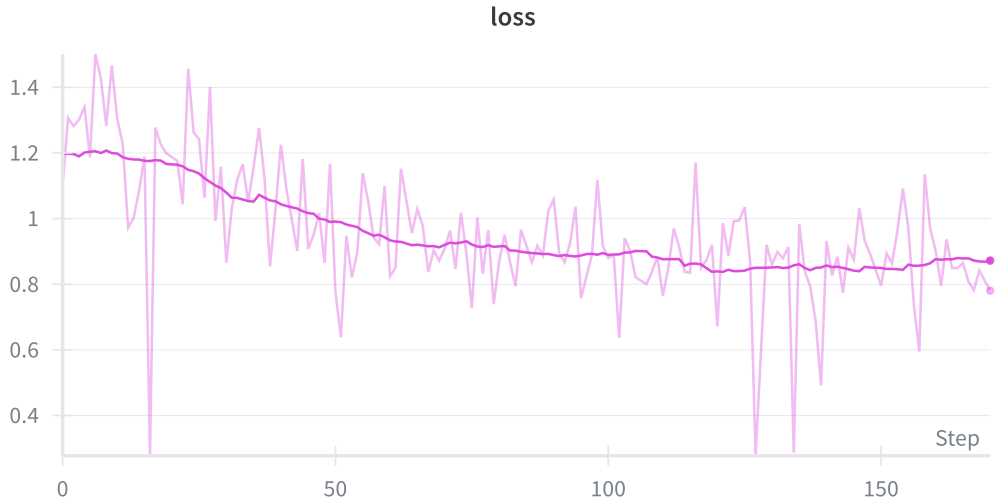

Figure 1: SFT training loss over epochs.

Reinforcement Learning (RL). Starting from the SFT checkpoint, we optimize using Group Relative Policy Optimization (GRPO), which computes advantages relative to the group mean and standard deviation across K rollouts per question. A judge model (Qwen3-8B-AWQ) evaluates each trajectory for correctness and approach quality (0–100). The reward combines sign from correctness with magnitude from approach:

$$R = \begin{cases} \frac{\text{approach}}{100} & \text{if correct} \\ -\left(1 - \frac{\text{approach}}{100}\right) & \text{if wrong} \end{cases} \tag{2}$$

This yields rewards in $[-1, 0]$ for wrong answers and $[0, 1]$ for correct answers.

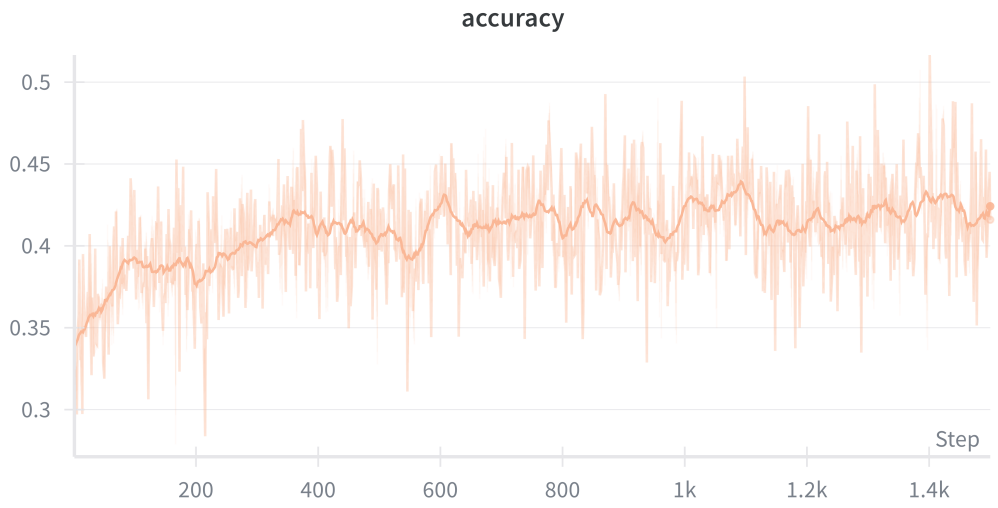

Figure 2: RL training accuracy over steps.

4.3 Hyperparameter Tuning

We set $K = 24$ rollouts per question to maximize GPU throughput on a 24GB GPU. Below this threshold, the GPU is underutilized; above it, VRAM pressure increases latency from ∼0.8s to ∼2s per rollout. Over 1,500 training steps (including judge inference), total training time was ∼23 hours; without this optimization it would have exceeded 50 hours. Larger K also reduces variance in advantage estimates. GRPO computes advantages relative to the group: for K rollouts with rewards $r_1, \ldots, r_K$:

$$A_i = \frac{r_i - \mu}{\sigma}, \quad \text{where } \mu = \frac{1}{K}\sum_{j=1}^{K} r_j, \quad \sigma = \sqrt{\frac{1}{K}\sum_{j=1}^{K}(r_j - \mu)^2} \tag{3}$$

We sample each rollout with temperature uniformly drawn from $[0.7, 1.15]$ for diversity. All rollouts share the same initial prompt, enabling batched generation.

5 Evidence of Meaningfulness

To verify that accuracy improvements reflect genuine search behavior rather than memorization, we inspected evaluation trajectories. We focused on whether correct answers came from executed search code versus lucky guessing. The following trajectories were selected as representative cases that exemplify the dominant behavioral patterns identified through qualitative analysis of evaluation outputs.

In successful trajectories, the model demonstrates clear information-seeking behavior: formulating targeted queries, iterating through results, and extracting answers from retrieved content. The following examples illustrate this pattern:

Ex 1: Helper function + keyword iteration

Q: Which acting school workshop did Linda Hamilton attend in New York City?

Agent:

```
def find_matching_section(content, word):
  idx = content.lower().find(word)
  if idx != -1: return content[idx-50:idx+100]
for r in search_pages("Linda Hamilton acting")[:5]:
  content = read_section(r['page_id']+":full")
  for w in ["workshop","school","training"]:
    match = find_matching_section(content, w)
    if match: print(match); break
```

Env:

```
attended acting workshops given by [Strasberg](Lee)(Lee
     Strasberg)
```

Agent: <answer>Lee Strasberg</answer>

Ex 2: Direct search + extraction

Q: What was the cause of Michael Clarke Duncan's death in 2012?

Agent:

```
for r in search_pages("Michael Clarke Duncan death cause
     ")[:5]:
  content = read_section(r['page_id']+":full")
  print(content); break
```

Env:

```
...On September 3, Duncan died in the hospital of
     respiratory failure. He was 54.
```

Agent: <answer>respiratory failure</answer>

These examples reveal several learned patterns. First, **query formulation**: the model constructs targeted search queries by combining the subject with relevant keywords ("Linda Hamilton acting", "Michael Clarke Duncan death cause"). Second, **iterative refinement**: Example 1 shows the model searching for multiple keywords ("workshop", "school", "training") rather than relying on a single query. Third, **programmatic extraction**: the model uses Python constructs—helper functions, loops, string slicing—to locate relevant context within retrieved pages. Fourth, **answer grounding**: in both cases, the final answer appears verbatim in the search output, confirming extraction rather than guessing.

Failed trajectories typically exhibit one of three patterns. **Interface misuse**: the model sometimes attempts to redefine the provided search functions rather than call them, causing syntax errors that waste the turn. **Silent execution**: other trajectories write elaborate filtering logic but forget to print results, leaving stdout empty and forcing a blind guess. **Hallucination under uncertainty**: when queries return irrelevant content, the model sometimes fabricates plausible-sounding answers rather than reformulating the query or acknowledging failure.

These findings suggest that the accuracy improvements reported in Section 4 reflect genuine learned behavior rather than memorization or luck. The model has acquired a repertoire of search strategies that generalize across different question types. The failure patterns are behavioral—misusing the interface, forgetting to print, giving up prematurely—rather than fundamental capability limits. This indicates that further training, particularly on recovering from failed searches, could yield additional improvements.

6 Conclusions

Our trained 4B parameter model achieves 42.68% accuracy on held-out questions, a 3.7× improvement over the 11.46% untrained baseline. SFT on expert trajectories establishes basic search patterns (28.13%), while RL adds 14.5 percentage points by optimizing for answer correctness. The key insight is that code generation provides more flexibility than fixed tool-use formats: the model learns to compose loops, conditionals, and string operations to navigate search results programmatically. Failure analysis shows most errors are behavioral—weak queries, premature termination, hallucination—rather than fundamental capability limits. Two directions for future work: training the model to detect retrieval failure and retry with reformulated queries, and evaluating on larger corpora beyond the 2,590-page dataset used here.