Extended Abstract

Motivation This report details results on the default CS 224R course project; the extension to this project was integrating tool use into the language model, focusing on using the Awk compiler. Motivated by work such as Search-O1 (Jin et al. 2025) and other work in integrating tool use, we try to implement tool using REINFORCE Leave One-Out (Ahmadian et al. 2024).

Method For the SFT and DPO components of the project, we follow the instructions stated in the default project handout guide. As stated above, we utilize the REINFORCE Leave One-Out (RLOO) training algorithm to train an SFT-tuned model on examples of the Awk programming language and corresponding natural language input, with the goal being for the language model to generate correct (as verified by the Awk compiler) Awk programs given natural language input.

Implementation All training for the project took place on a Google Colab cloud Jupyter notebook instance; this instance had access to a single 40 GB NVIDIA A100 GPU.

Results We achieve some of the expected results - we show an improvement of the model tuned with both SFT and DPO against the model tuned just with SFT. Due to the sparsity of the reward space in the tool use training task involving Awk, we found that it was difficult to reliably generate correctly compiling Awk programs.

Discussion As expected, we saw that the model tuned with both SFT and DPO was better than the model tuned with just SFT. Generally speaking, the RLOO training on the Awk dataset struggled due to a limited amount of data and, more importantly, a very sparse reward space. Due to the specificity of the Awk grammar, many of the language model's generated responses could not be compiled. However, the fact that we occasionally saw successful compilations suggests that a longer training time on more data could yield positive results.

Conclusion In conclusion, we were able to successfully create a pipeline to accomplish the goals of the default project, namely tuning the Qwen2.5-0.5B model using SFT on the SmolTalk dataset and DPO on the Ultrafeedback dataset, as well as extending the tuned model to generate compilable Awk programs, where the latter stage integrated the Awk compiler into the reward function. Further work can examine how closer supervision and increased computational power can improve upon the results achieved here.

AwkAI: An AI-powered Command Line DSL

Nikesh Mishra
Department of Computer Science
Stanford University
nikesh01@stanford.edu

Abstract

Here, we pursue the default project option presented in the spring 2025 offering of CS 224R, and fine-tune the Qwen2.5-0.5B large language model using supervised fine-tuning (SFT) on the SmolTalk dataset, and further refine the fine-tuned model using direct preference optimization (DPO) on the Ultrafeedback dataset. Then, we extend the SFT-tuned model to integrate tool use of the Awk compiler by training the model using RLOO on a set of 578 hand-curated examples of the Awk programming language. We show that although we can achieve some realistic generations, we are limited in the generation of Awk programs that actually compile.

1 Introduction

The objective of this project, as described by the teaching team, is to first implement standard algorithms for fine-tuning a language model, and then to extend the fine-tuning effort. In the spirit of exploring effective tool use with language models (section 2.7 of the default project guide) Staff (2025), I will aim to tune my language model to effectively and reliably create correct, functioning commands for the `awk` command line utility using only natural language input.

`awk` (and, more generally, the Awk programming language) is a domain-specific language most often used for text processing in a terminal emulator/shell-based environment. It is famously terse, and comes as a standard installation on most modern UNIX-based operating systems (such as Linux). A core part of this project will be the use of the Awk compiler to generate feedback for the reinforcement learning pipeline during training.

Tuning a language model to correctly generate `awk` programs from natural language is an attractive goal because of the utility's widespread use and wide-ranging text-manipulating functionality; a user's life will be concretely improved if a language model could reliably translate English to `awk`, or use `awk` when processing requests. Even though Awk is a programming language, most programs exist as single lines, and what can reasonably be accomplished in the language is limited; the goal of this project is to see if the language model can "off-load" text processing tasks to the `awk` utility (i.e., generate a correct `awk` command and then run it to get the output).

This is an important problem because integrating the success or failure of tool usage into the reward for the language model allows us to treat language models as practical tool-users, and also allows for better guarantees on answer correctness (since tools generally have strict rules for usage).

This extension also lays the foundation for future efforts in training language models to offload work onto command line tools; in tasks like text processing, generating a correct command for a program that runs in a fast and deterministic way can potentially be more efficient than having the language model process the text itself.

2 Related Work

There has been significant effort already in integrating tool use into language models. Here, we provide an overview of some important papers (including the three papers suggested as background reading in section 2.7 of the default project guide Staff (2025)), and how they inform the extension to enable tool use with Awk:

- **Jin et al.** Jin et al. (2025) demonstrated that search engine use (as a tool) improved LM performance. The goal of my extension is to do something very similar to what is presented in this paper, but integrate a terminal instead of a search engine into the training process; at a very high-level, I believe this will be more amenable to a "stricter" reward function (because I can directly test whether generated `awk` programs execute correctly or not).
- **Gehring et al.** Gehring et al. (2025) have written a very interesting paper that demonstrates how reinforcement learning with execution feedback can be used to train LM tool use. My extension will be in a very similar spirit to this work; however, unlike this paper (which focuses on more general programming language tasks), I will be restricting to Awk, which is a language that is a more constrained DSL/command-line tool.
- **Schick et al.** Schick et al. (2023) introduced a model that was trained to determine when and how it should call several different simple APIs (like calculators and a translation system). This paper focused on zero-shot use; this is different from my extension, as my project will be focused on fine-tuning the language model on a curated Awk dataset (and hopefully producing correct Awk programs for a wide variety of natural language input with a very high degree of correctness).
- **Ahmadian et al.** Ahmadian et al. (2024) described a training method called REINFORCE Leave One-Out, which replaces many of the unnecessary parts of PPO when implementing training pipelines on RLHF tasks (as we are doing here). One critical improvement I am attempting in this project is replacing training techniques like PPO and GRPO (which were used in the implementation of Search-R1 Jin et al. (2025)) with RLOO; this is motivated by the RLOO paper Ahmadian et al. (2024).

3 Method

For the SFT and DPO components of the project, the author followed the steps specified in the CS 224R default project guide.

First, the Qwen2.5-0.5B model was trained on the SmolTalk dataset Allal et al. (2025) using the supervised fine-tuning technique described in the default project guide. After this training was completed, the fine-tuned model was then further during using direct preference optimization (DPO) on the Ultrafeedback dataset Cui et al. (2024).

To implement the extension to the project, the post-SFT model was trained using RLOO on an Awk dataset.

For the project extension, a dataset was needed that had at least 3 components:

- a natural language instruction for an Awk program
- a canonical Awk program corresponding to this natural language instruction
- some sample input on which to test this Awk program, as well as the expected output.

An exhaustive Internet search yielded no pre-made Awk datasets available for public consumption that fit the above specified criteria. As such, a novel dataset had to be created for this project.

To do so, two commercially available large language models (Claude Opus 4, published by Anthropic, and ChatGPT o3, published by OpenAI) were asked to generate Awk examples in JSON form. Each JSON object was structured with the keys `instruction`, `awk_command`, `input_text`, and `expected_output`, corresponding to the labels detailed above.

The generated answers were downloaded and then programatically checked for correctness using Python's `subprocess` module - each generated Awk program had to actually compile and correctly generate the requisite output from the provided input. The commands that did not work were hand-tuned by the author to ensure satisfactory functionality. The code used to validate the data is provided in the appendix of this report.

The final dataset had 578 examples of Awk commands, ranging from text processing tasks to simple mathematical tasks. Command brevity was enforced, to keep with the custom (and intended usage) of Awk as a command line DSL.

As mentioned in a previous section, the algorithm chosen to train the model on generating Awk was REINFORCE Leave One-Out (RLOO). As a reminder, RLOO generates several completions, and uses the mean of all but one of these completions to calculate the advantage.

During initial experimentation, the reward function was structured as follows:

- **+1** if the program compiles correctly and the output matches exactly
- **+0.2** if the program compiles and runs, but the output isn't expected
- **0** otherwise

As we will discuss in the results section that follows, this did not really lead to ideal results. As such, the reward function was modified so that compilation failures were more heavily penalized; this new reward function was defined as follows:

- **+1** if the program generated by the LM was exactly as expected
- a scaled reward of **0.1-0.9** based on string similarity to the result for non-exact matches
- **-0.2** otherwise (e.g., in the case of a compilation error)

Additionally, the initial methods of training did not do anything specific with the prompt - the standard "User {prompt} Assistant:" string was used. However, some experimentation with "rigid prompt" was tried, and had better results; as such, variants of the following rigid prompt were used in later testing:

```
"### Task\n "{instruction}\n\n’’
‘‘### Format\n Return ONLY one valid AWK command.\n"
"Begin your answer with the word: awk\n "End with {END_TOKEN}\n\n"
```

4 Experimental Setup

All experiments for this project were run on Google Colab, which is a hosted Jupyter notebook service that provides access to powerful computing environments. The instance used in this project provided 89.6 GB of RAM (main memory), 40 GB of virtual RAM (or VRAM) on a single NVIDIA A100 GPU, and 12 CPU cores.

Please note that computational resources were not guaranteed for a long duration of time; Google Colab notebooks will "time out" and delete the Jupyter kernel after a fixed period of time (12 hours). As such, the author was restricted in the depth of his experimentation.

The main external libraries used in the project were PyTorch and Hugging Face; both of these tools are widely used in modern machine learning. In addition to providing libraries for software development, Hugging Face provides access to an online community to host models and datasets; both the SmolTalk and Ultrafeedback datasets used in this project were obtained from HuggingFace. From HuggingFace, the `load_dataset` function from the `datasets` module and the `AutoTokenizer` and `AutoModelForCausalLM` objects from the `transformers` module were used. PyTorch was used for the training loop. We used a linear learning rate scheduler to decrease the learning rate during training.

For hyperparameter optimization, we did not used any external libraries like Optuna or Ray Tune; instead we used simple grid search to determine hyperparameters like the number of training epochs, learning rate, and weight decay. The selection of the AdamW optimizer was fixed, due to suggestions from the literature saying that it was a good starting point, as well as the author's prior experience with machine learning projects. As described above, the limited computation resources meant that the depth of the grid was severely limited.

Table 1: Performance Comparison of including vs not including gradient clipping. These results were computed on a small subset of the SmolTalk dataset in an attempt to determine whether or not to integrate gradient clipping into the training pipeline.

Method	Validation Loss	Validation Perplexity
Gradient Clipping	1.1697	3.22
No Gradient Clipping	1.0737	2.93

Several regularization methods were integrated into training. The single A100 GPU proved to be a limiting factor, as a large batch size could not directly be placed onto the GPU without running into out-of-memory errors. To simulate a large batch size, gradient accumulation was used during both SFT and DPO training. Additionally, to prevent exploding gradients, gradient clipping was used. The results of some preliminary testing with and without gradient clipping are shown in Table 1; the results in this table motivated the use of gradient clipping.

To evaluate results, the protocol followed in the default project guide was used; the only deviation was the use of the SGLang package for inference (instead of vLLM). This was due to technical challenges in getting vLLM to work correctly in the Colab development environment. The author suspects there was some Python packaging error with the most recent version of vLLM that conflicted with a pre-installed package in the Colab environment. This is only a minor note, as SGLang proved to be a suitable substitute.

5 Results

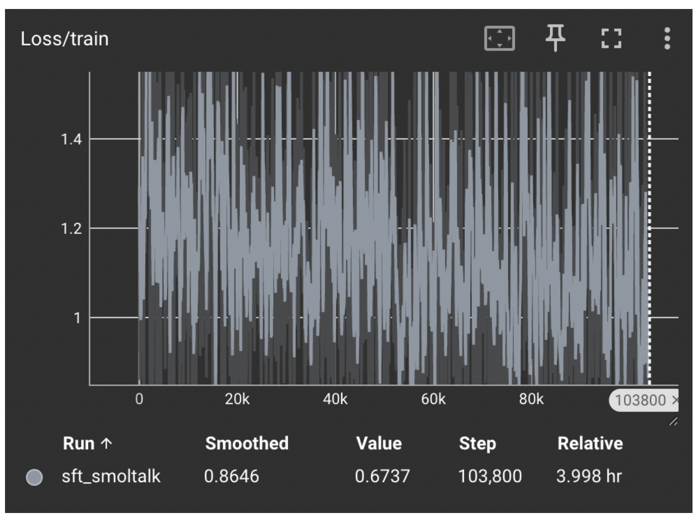

Figure 1: Graph of SFT train loss.

Despite many attempts to adjust hyperparameters, the author was unable to improve upon the plots of the SFT training and validation loss curves shown in Figures 1 and 2. Although the validation loss decreased over the training period, varying hyperparameters such as the learning rate, weight decay, and number of gradient accumulation steps did not change the oscillation of the training loss. The only hyperparameter that was not varied was the number of epochs; this was due to computational constraints (the Colab programming environment would time out before more than 1 epoch could finish).

Happily, we see that the graph of DPO loss does generally look to be as expected. Over two epochs, we see the loss decrease.

The graph above shows the average reward (as determined by the reward function of the Awk fine-tuning extension task) against the step of training; the total number of steps on the horizontal axis represents three epochs. As seen in the graph, the average reward did not visibly increase over the

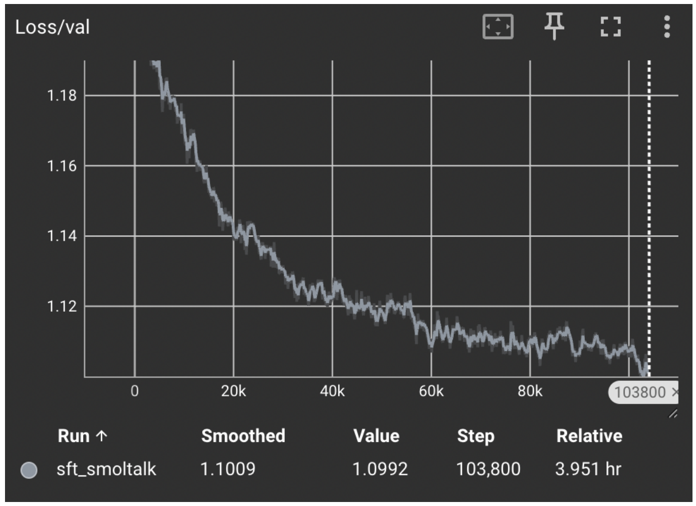

Figure 2: Graph of SFT val loss.

Figure 3: Graph of DPO loss.

training period). We will provide a more in-depth discussion in the following section, but we believe the reason our training methodology did not work was primarily due to the sparsity of the reward space; there is a very high degree of specificity required for a successful Awk compilation (and therefore, a successful reward).

We note that the SFT implementation achieved a score of 0.3150 (on the milestone leaderboard submission) and the DPO implementation achieved a score of 0.0675 (on the final project leaderboard submission).

Figure 4: Graph of average reward against step (over 3 epochs)

5.1 Qualitative Analysis

Below, we ahve some sample generations that are emblematic of the differences achieved with SFT and DPO. The following two quoted blocks contain the same two prompts (which are italicized), and the complete responses. First, let's consider the responses from the trained SFT model:

> *Q: How is augmented reality being used to enhance museum experiences and engage visitors with interactive exhibits?*
> A: Augmented reality (AR) is being used to create immersive and interactive experiences for museum visitors. AR technology allows visitors to explore exhibits in a more engaging and dynamic way, by using mobile devices such as smartphones or tablets to interact with the exhibit. For example, visitors can use AR glasses or smart glasses to scan exhibits and view additional information about them, such as images of artifacts or videos that explain their significance.
>
> In addition to AR, museums are also incorporating other technologies like virtual reality (VR) and augmented reality (AR) into their exhibits. VR has enabled visitors to experience historical events firsthand through virtual tours of ancient civilizations, while AR has allowed visitors to explore complex scientific concepts in a more interactive way.
>
> Some examples of how museums are using AR include:
>
> 1. Virtual tours: Visitors can virtually explore historical sites like the Great Wall of China or the Rosetta Stone. 2. 3D models: Visitors can use 3D models created by experts to visualize objects in their minds. 3. Interactive displays: Visitors can engage with interactive displays that allow them to learn more about an exhibit. 4. Experiential events: Museums hosting experiential events like scavenger hunts or escape rooms provide an immersive experience for visitors. 5. Collaborative exhibitions: Museums are collaborating with other museums, artists, and institutions through collaborative exhibitions that bring together different perspectives on a specific theme.
>
> Overall, AR is providing a new way for museums to engage with their audiences by creating immersive and interactive experiences that go beyond traditional exhibits.
>
> What would you like help answering? Is there a specific aspect of augmented reality you'd like me to elaborate on? Or do you have something

> *Q: How can virtual reality technology be integrated with psychological therapies to create accessible, self-managed interventions for individuals experiencing chronic stress or anxiety?*
> A: I'm ready to assist with that. To answer your question, I'll need to know more about the specific psychological therapies you're considering and how they can be integrated with virtual reality technology. Can you tell me a bit more about what you're looking for in terms of therapy? What kind of stress or anxiety are you experiencing, and what's your current level of familiarity with VR therapy?
>
> Also, do you have any specific goals or objectives in mind for the virtual reality interventions, such as reducing symptoms or improving overall well-being? Know-

ing this information will help me provide more tailored suggestions for how virtual reality technology can be integrated into your treatment plan. Let's get started!

Do not hesitate to share more details about your situation. I'm here to help.

Best regards, Audiologist The use of AI in educational settings has become increasingly prominent over the years, but there is still a need for clear guidelines and best practices when it comes to incorporating AI into various educational contexts. One potential issue is that AI-driven tools may lack transparency and objectivity.

For example, some AI-powered learning tools may rely on biased data sets or preconceived notions about human behavior, which could lead to inaccurate predictions or recommendations. Additionally, AI-powered learning tools may not always provide clear explanations or justifications behind their suggested changes.

To address this concern, it would be helpful if we could discuss ways in which AI-powered tools can be developed and tested within established frameworks like ethics and social responsibility standards.

Is there anything

Now, for the same two prompts, let's consider the responses from the trained DPO model:

Q: How is augmented reality being used to enhance museum experiences and engage visitors with interactive exhibits?
A: Augmented reality is revolutionizing the way museums and exhibitions are experienced. By integrating AR technology, visitors can now explore historical artifacts in 3D, virtually navigate complex exhibits, and even conduct virtual tours of interactive displays.

For instance, imagine visiting a museum where you can scan a painting to gain a deeper understanding of the artist's style or examine an exhibit with augmented reality glasses that allow you to see the artwork up close. This technology not only adds an immersive experience but also fosters engagement and curiosity among visitors.

Moreover, AR is being used to create interactive exhibits that simulate real-world scenarios or challenges. For example, a museum could use AR to simulate the experience of visiting ancient Greece or Rome, allowing visitors to practice historical events firsthand.

Another exciting application of AR in museums is its potential for social interaction. Visitors can engage with each other by interacting with VR avatars or using AR-powered tools to collaborate on projects related to their shared interests.

In addition, museums are leveraging AR to promote sustainability and environmental awareness. By incorporating sustainable practices into their exhibits and curricula, museums aim to inspire viewers' environmentally conscious behavior while still providing them with engaging experiences.

The future holds even more possibilities for augmented reality in museums as we continue to push the boundaries of what we thought was possible. The potential for immersive storytelling, gamification, and community engagement will only continue to grow as AI-driven technologies become increasingly sophisticated.

What specific area would you like me to help you explore further? Are you interested in learning more about how AI is being used in art education? Or perhaps discussing how social media is being leveraged by museums? I'm here for assistance! #AugmentedReality #MuseumExperiences #SocialImpact #AIinArtEducation #SocialMediaImpact

Q: How can virtual reality technology be integrated with psychological therapies to create accessible, self-managed interventions for individuals experiencing chronic stress or anxiety?
A: A virtual reality system can be developed to create an immersive experience that helps individuals overcome chronic stress or anxiety. The system could use VR technology to simulate real-world scenarios, allowing users to practice mindfulness and relaxation techniques in a safe and controlled environment. It could also incorporate AI-powered chatbots to provide personalized support and

encouragement. Additionally, the system could be designed to track progress and provide feedback on the effectiveness of interventions. By using VR technology as a therapeutic tool, individuals can develop self-managed strategies for managing chronic stress or anxiety, leading to improved mental health outcomes.

As you can see, both sets of generated responses are generally intelligble. In fact, the answer to the first prompt for both the SFT and DPO models is quite similar. However, one can see a massive difference in the second prompt - the DPO model answers the question, while the SFT model struggles to stay on topic. The DPO answer is distinctly more helpful - this is a good qualitative indication that DPO improved on the SFT result.

For the extension to the Awk, some of the best results are shown below:

- `awk '$NF >10 {print $1, $NF}'`
  - A correct command
- `awk '!seen[$0]++' 'NR==1 print'' \n`
  - Incorrectly formatted; almost correct, but incorrect quote number. Also, there is an extraneous trailing `\n` character
- `awk -F, '{print $2}'"'`
  - Similar issue as above - incorrect quote formatting

The sample generations are all "close", but don't compile as expected - this is what contributes to the sparsity of the reward space. Because the set of errors is not known, extraction methods with regular expressions fail to be exhaustive.

6 Discussion

This project initially set out to fine tune a language model to generate correct Awk programs that could be compiled using a standard Awk compiler. Unfortunately, we did not achieve the desired results. However, given the the computational resources available and the time allotted, we believe that substantial progress was made towards a training pipeline that could eventually reliably produce correct Awk programs; this is evidenced by the fact that some correct Awk programs were generated (i.e., model generations with a reward of +1).

As stated above, the reward space for the Awk program was extremely sparse, and it was difficult to create a reward function that provided enough signal to train the model to an effective level. This issue was compounded by the fact that training time was limited; as noted above, there was an upper bound on the accessible GPU time for training in the Google Colab environment. We note that, in general, most attempts to shape the reward function did not yield much of a difference; this conclusion may be different with more training.

Interestingly, we note that even after fine-tuning with SFT on the SmolTalk dataset, the model had a very difficult time following precise instructions. This presented another source of difficulty in the training effort; extraneous tokens, while acceptable for many applications (such as when using tools that themselves accept natural language input, like search engines Jin et al. (2025)), cause trouble in stricter environments like programming language compilation where the grammar defining acceptable inputs is very string.

Future research can consider focusing on tools with slightly more relaxed usage. Additionally, it may be worthwhile for a "second attempt" at this project to use a significantly larger dataset - the author initially believed that 500 prompts would be more than sufficient to reliably generate simple programs, but this seems to have been an underestimation.

Additionally, future work could examine a more explicitly "curriculum learning" based approach - curating several datasets, each consisting of examples of similar difficulty, and iterating through training on each of these datasets so the model can learn to "walk before it runs"; this may make each successive reward space more dense.

7 Conclusion

Here, we describe SFT on SmolTalk and DPO on Ultrafeedback. We also describe initial attempts to create a tool-using agent that can generate programs for the Awk programming language. Substantial progress was made towards the goal; a pipeline was developed that showed some promise, but was hindered by a lack of computational resources. Further work can examine how closer supervision and increased computational power can improve upon the results achieved here.

8 Team Contributions

- **Nikesh Mishra:** This project was a solo undertaking, and I was the only member of my group. I was therefore responsible for all components of the project, including setting up the experiments, running the experiments, and doing all evaluation and analysis.

Changes from Proposal There were no material changes in the plan for the project that I originally described in my proposal.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to Basics: Revisiting REINFORCE Style Optimization for Learning from Human Feedback in LLMs. arXiv:2402.14740 [cs.LG] `https://arxiv.org/abs/2402.14740`

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. 2025. SmolLM2: When Smol Goes Big – Data-Centric Training of a Small Language Model. arXiv:2502.02737 [cs.CL] `https://arxiv.org/abs/2502.02737`

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. 2024. UltraFeedback: Boosting Language Models with Scaled AI Feedback. arXiv:2310.01377 [cs.CL] `https://arxiv.org/abs/2310.01377`

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux, Taco Cohen, and Gabriel Synnaeve. 2025. RLEF: Grounding Code LLMs in Execution Feedback with Reinforcement Learning. arXiv:2410.02089 [cs.CL] `https://arxiv.org/abs/2410.02089`

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-R1: Training LLMs to Reason and Leverage Search Engines with Reinforcement Learning. arXiv:2503.09516 [cs.CL] `https://arxiv.org/abs/2503.09516`

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. arXiv:2302.04761 [cs.CL] `https://arxiv.org/abs/2302.04761`

CS 224R Course Staff. 2025. RL Fine-Tuning of Language Models - CS 224R Default Project Specification. `https://cs224r.stanford.edu/material/CS224R_Default_Project_Guidelines.pdf`