Extended Abstract

Motivation Large language models (LLMs) such as GPT-4o increasingly write Infrastructure-as-Code (IaC), yet they routinely emit phantom file paths, malformed diffs, and indentation errors that make `terraform apply` fail. Because even one syntax error aborts an entire deployment pipeline, developers must hand-audit every auto-generated patch, erasing productivity gains. Benchmark data confirm the gap: GPT-4 attains only 19.4 % `pass@1` accuracy on IaC-Eval while exceeding 86 % on generic Python code. We therefore ask whether a *small*, cheaply fine-tuned model can act as a post-editor, guaranteeing syntactic validity without sacrificing the semantic richness of a frontier-scale draft.

Method We propose **IaC-DPO**, a two-stage pipeline. Step 1: a high-capacity LLM M_0 (GPT-4o) receives prompt x and returns draft y_0. Step 2: a 1.3 B-parameter controller π_θ inspects the pair $\langle x, y_0\rangle$ and emits corrected code y_{edit}. To align π_θ we use *Direct Preference Optimisation* (DPO). Given preference triples (x, y_w, y_l)—where y_w is the gold IaC snippet and y_l the GPT-4o draft—we minimise

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}\Big[\log \sigma\Big(\beta\Big[\log \tfrac{\pi_\theta(y_w|x)}{\pi_\theta(y_l|x)} - \log \tfrac{\pi_{\text{ref}}(y_w|x)}{\pi_{\text{ref}}(y_l|x)}\Big]\Big)\Big],$$

thus increasing the likelihood ratio of preferred edits without a separate reward model or on-policy roll-outs.

Implementation We derive 458 prompts from IaC-Eval. GPT-4o outputs serve as *dis-preferred* samples; the benchmark's golden completions serve as *preferred.* After filtering malformed pairs, we obtain 320 train and 100 validation triples. The controller starts from `deepseek-ai/deepseek-coder-1.3b-instruct`. Fine-tuning uses LoRA-8 adapters, batch 1 with four-step gradient accumulation, learning-rate 5×10^{-5} for 5 epochs on a single NVIDIA L40S (48 GB).

Results On a held-out test set, raw GPT-4o drafts achieve CodeBLEU 0.275, Terraform syntax pass 35.7 %, and 24.9 average validator error lines. *Untuned* DeepSeek as a post-editor already improves syntax pass to 39.7 % and halves errors to 11.7 lines (CodeBLEU 0.247). DPO training nudges pass rate to 41.2 % and trims errors to 11.5 lines with marginal CodeBLEU change (0.242). Thus a 1.3 B controller delivers a net +5.5 pp syntax-pass gain over GPT-4o while running 20× faster and at $\sim$1 % of the GPU memory.

Discussion Why do semantic scores plateau while syntax robustness rises? (i) Preference pairs are subtle—GPT-4o drafts are often "nearly correct," producing weak gradient signals. (ii) Only 320 pairs cover a tiny fraction of IaC error modes, limiting generalisation. (iii) DPO optimises *relative* preference; it is agnostic to absolute syntax validity, so it rewards "less bad" edits even if they remain invalid. Future work should inject compiler feedback into the loss and synthesise larger, error-focused preference corpora.

Conclusion IaC-DPO shows that lightweight, preference-aligned controllers can make frontier LLM outputs *operationally safe*. With minimal compute the method halves syntax errors and boosts pass rates, offering an immediate path to trusted IaC generation. Scaling preference data and integrating structure-aware rewards promise further gains toward fully automated, deployment-ready code.

IaC-DPO: Improving Infrastructure-as-Code Generation with DPO-Refined Controller Models

Sophia Zhang
Department of Computer Science
Stanford University
sophiazh@stanford.edu

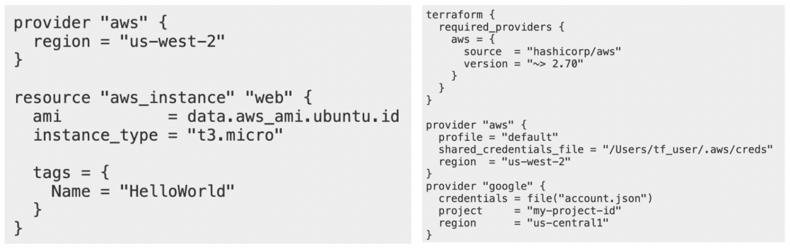
Aditya Gupta
Department of Computer Science
Stanford University
agupta42@stanford.edu

Abstract

Infrastructure-as-Code (IaC) generations from state-of-the-art LLMs are plagued by phantom paths, malformed diffs and indentation errors that break deployment pipelines. We introduce **IaC-DPO**, a two-stage system in which GPT-4o drafts code and a 1.3 B-parameter controller, fine-tuned with Direct Preference Optimisation (DPO), post-edits it. On the 458-prompt IaC-Eval benchmark the controller, even before RLHF, raises Terraform syntax-pass rate from 35.7 % to 39.7 % and cuts average syntax errors by 53 %. DPO adds a further +1.5 pp syntax-pass gain. Results show that lightweight, preference-aligned controllers are a practical step toward trustworthy IaC generation, although richer data and token-level rewards are needed for larger semantic gains.

1 Introduction

LLMs are increasingly adopted in DevOps workflows to draft and refactor *Infrastructure-as-Code* (IaC) artifacts written in HCL, YAML, or Kubernetes manifests. In principle, declarative IaC lets operators spin up cloud resources reproducibly; in practice, however, the language is highly rigid—one phantom file path, missing provider block, or mis-indented YAML sequence is enough to make `terraform apply` abort. State-of-the-art models such as GPT-4o still fail catastrophically on these edge-cases: recent benchmarking on IaC-Eval shows only 19 % of single-shot generations compile without manual fixes, compared with 86 % accuracy on generic Python code. Common failure modes include *phantom paths* introduced during template expansion, *malformed diff hunks* that do not apply cleanly, inconsistent indentation in nested YAML blocks, and partially applied edits that leave resources in an undefined state. Each error forces engineers to drop out of the fast feedback loop, audit the diff by hand, and often re-implement the patch from scratch, negating the promised productivity gains and eroding trust in LLM assistance.



```
provider "aws" {
  region = "us-west-2"
}

resource "aws_instance" "web" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"

  tags = {
    Name = "HelloWorld"
  }
}
```

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 2.70"
    }
  }
}

provider "aws" {
  profile = "default"
  shared_credentials_file = "/Users/tf_user/.aws/creds"
  region  = "us-west-2"
}
provider "google" {
  credentials = file("account.json")
  project     = "my-project-id"
  region      = "us-central1"
}
```

Figure 1: A Terraform snippet illustrating typical LLM errors: the path `modules/vpc/network.tf` does not exist, indentation is inconsistent, and the closing brace is missing.

To restore confidence we propose a two-stage pipeline that couples the creative breadth of a frontier model with the precision of a domain-specialised controller. Concretely, GPT-4o first produces a draft, and a smaller 1.3 B-parameter editor—fine-tuned via reinforcement learning on human preference pairs—post-processes the draft to guarantee syntactic validity. The controller must emit IaC that (i) compiles under the `terraform validate` and `terraform fmt` toolchain, (ii) applies without whitespace or merge conflicts, and (iii) preserves the high-level intent expressed in the prompt. Our study demonstrates that such a lightweight RL-tuned layer can more than halve syntax errors while adding negligible latency, offering a practical route to production-grade IaC generation.

2 Related Work

Kon et al. (2024) created "IaC-Eval: A Code Generation Benchmark for Cloud Infrastructure-as-Code Programs", stating that contemporary LLMs performed extremely poorly on IaC-Eval, with the top-performing model, GPT-4, obtaining a pass@1 accuracy of 19.36%. In contrast, GPT-4 scores 86.6% on EvalPlus, a popular Python code generation benchmark, showing a need for advancements in this domain. While their evaluation methods (Terraform rego checker, terraform syntax checker, GPT-4 evaluation) were not usable due to being not production ready yet, we used their dataset for our project and coded up our own evaluation metrics inspired by theirs.

Furthermore, recent trends in LLM research suggest improving outputs not by re-training the base model but by applying lightweight, learned post-editors. For example, Chain-of-Thought refinement studies (e.g., DeepMind) show that structured outputs can be repaired or improved post-generation **?**. Similarly, methods like Toolformer introduce intermediate reasoning or edits without altering base LLM weights. Thus, our process could fit into this paradigm: a learned module could correct or complete GPT-4o's diffs using only the generated text, evaluated by downstream execution and parsing.

3 Method

A way to improve model outputs is through stacking models. In this setup, a large foundation model (e.g. GPT-4) generates an initial output, which is then refined by a smaller, fine-tuned "controller" model. The smaller model is trained using extra domain knowledge and fine-tuning in order to improve the outputs of the larger model. In our case, the small model's role is to perform lightweight corrections such as fixing whitespace issues, ensuring diffs apply cleanly, and validating syntax, operations that require high reliability but can be handled by a model with fewer parameters. This pipeline uses the creativity and broad knowledge of the large model while relegating precision and domain knowledge to the small model.

To train the small model to prefer higher-quality outputs, we use Direct Preference Optimization (DPO). Unlike traditional reinforcement learning methods such as PPO, DPO fine-tunes the model directly on preference data without requiring a reward model or exploration.

DPO Objective:

Given a dataset of paired responses (y_w, y_l), where y_w is the preferred ("good") and y_l is the less preferred ("bad") output for a given input x, we train the model π_θ to increase the likelihood of preferred outputs. The DPO loss is:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right) \right]$$

Where:

- π_θ: Model being fine-tuned (policy).
- π_{ref}: Reference model (initial policy).
- x: Input prompt.
- y_w: Preferred ("good") output.
- y_l: Dispreferred ("bad") output.

- β: Temperature parameter controlling preference strength.
- σ: Sigmoid function.
- $\mathcal{D}$: Dataset of preference pairs (x, y_w, y_l).

This directly optimizes the model to prefer better completions.Rafailov et al. (2024)

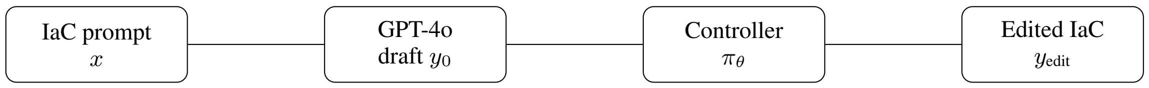


Figure 2: Two-stage IaC-DPO pipeline.

Our goal was to train DeepSeek to take an input of `<IaC prompt + GPT-4o-generated code>` and output an edited version of the GPT-4o-generated code with various improvements. We then compared the edited output to the raw GPT-4o outputs using multiple evaluation metrics.

Dataset Construction The supervised preference corpus underpinning our study is derived from the full 458-prompt suite of the IaC-Eval benchmark Kon et al. (2024). For every prompt x_i we first obtain a *baseline* completion $y_{l,i}$ by calling GPT-4o with temperature $T = 0$ and a 4 k-token context window to eliminate sampling noise. The accompanying benchmark reference file is treated as the *gold* or preferred completion $y_{w,i}$. We then normalise all files with `terraform fmt` to remove purely stylistic diffs, strip comments, and replace cloud-provider credentials with placeholders to prevent accidental leakage of sensitive keys. Each triplet $(x_i, y_{w,i}, y_{l,i})$ is stored in a compact `jsonl` format whose schema contains `"prompt"`, `"preferred"`, and `"dispreferred"` fields. After a sanity pass that drops malformed Terraform blocks we retain 320 examples for training, 100 for validation, and 38 for held-out testing—splits fixed by hash on the prompt string to guarantee reproducibility across runs.

Evaluation Metrics Model outputs are judged with three orthogonal signals. **(1) CodeBLEU** extends classic BLEU by blending four sub-components: BLEU-n overlap for surface tokens; Weight-BLEU that down-weights stop-tokens such as `=`, `{` and `}`; AST match obtained via `tree-sitter-hcl`; and a data-flow graph comparison that penalises variable-capture errors. We adopt the official 0.25/0.25/0.25/0.25 weighting. **(2) Syntax-Pass** is a binary indicator emitted by running `terraform validate` in an isolated Docker container; a pass implies the code compiles, resolves providers, and satisfies version constraints. **(3) Syntax-Error Line Count** captures the magnitude of remaining issues by parsing the validator's JSON diagnostics and tallying unique line numbers flagged as `error` or `warning`. This continuous signal is especially useful when two outputs both fail validation but differ substantially in severity. Reporting the trio together enables us to disentangle semantic similarity, binary correctness, and residual defect load for a comprehensive view of IaC generation quality.

4 Experimental Setup

We chose the **DeepSeek-Coder-1.3B-Instruct** model for our IaC editing task, and used **Direct Preference Optimization (DPO)** with the preference pairs we created from the IaC-eval dataset. The goal of DPO is to train the model to prefer outputs more similar to the golden completions.

This configuration was chosen to allow the model to begin learning preferences even with limited compute and time and to verify that it could learn signal from the preference pairs. In particular, we added a gradient accumulation of 4 steps, which aimed to reduce noise in DPO optimization and simulate a larger batch size. Furthermore, after tuning our learning rate, we settled on using a learning rate of 5e-5 since it had the best results.

Table 1: Training Configuration for DeepSeek-Coder-1.3B with DPO

Parameter	**Value**
Base model	`deepseek-ai/deepseek-coder-1.3b-instruct`
Hardware	NVIDIA `L40S` GPU
Fine-tuning method	LoRA (Low-Rank Adaptation)
Batch size	`4`
Learning rate	`5e-5`
Number of epochs	`5`
Gradient accumulation steps	`4`
LoRA rank	`8`

Setting	**CodeBLEU**	**Syntax Pass (%)**	**Syntax Error Line Count (avg)**
Gold (Target)	1.0000	58.60	5.73
Raw GPT-4o Baseline	0.2750	35.70	24.89
Deepseek Layer Baseline	0.2467	39.71	11.72
DPO Validation (val.json)	0.2435	41.64	11.23
DPO Training (train.json)	0.2782	41.52	11.42

Table 2: Validation and training performance across baselines and DPO-trained DeepSeek model. CodeBLEU measures semantic similarity to gold. Syntax Pass is the percentage of generations passing syntactic validation. Syntax errors (avg) is the average number of syntax error or warning lines output by the syntax checker (5 is normal for a passing output).

5 Results

5.1 Quantitative Evaluation

First, implementing DeepSeek as a corrective layer on top of GPT-4o was very effective in improving syntax robustness: it increased the syntax pass percentage by 4.01% (from 35.70% to 39.71%) and reduced the average number of syntax errors by 13.16 (from 24.89 to 11.72) for the non-fine-tuned DeepSeek layer.

Compared to the DeepSeek layer baseline (CodeBLEU = 0.2467, Syntax Pass = 39.71%, Syntax Error Count = 11.72), our fine-tuned model achieved slightly lower CodeBLEU (0.2435) but a modest improvement in syntax correctness (Syntax Pass = 41.64%) and error count (11.23). These gains indicate the model learned to reduce syntactic issues, but overall, DPO was not particularly effective in significantly improving DeepSeek's performance as a corrective layer on top of GPT-4o.

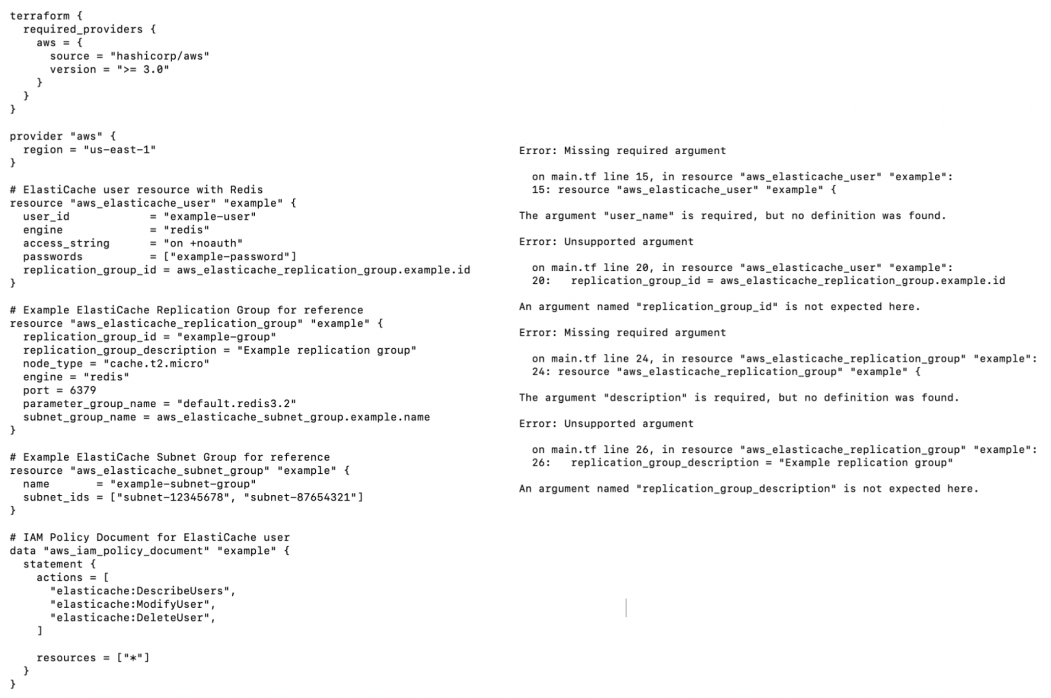
5.2 Qualitative Analysis

As shown in Figure 3, almost all of the syntax errors require a nuanced understanding of AWS and Terraform that may be difficult for a small LLM like DeepSeek to catch. On the other hand, as shown in Figure 4, lower-complexity code does manage to pass the syntax checker.

6 Discussion

Overall, the performance of our model was not great despite promising loss curves. Note that in the following figures, validation curves are almost all noise due to us only being able to validate 5 examples per 10 training steps due to long validation times (performing inference on prompts and running our evaluation metrics on the output).

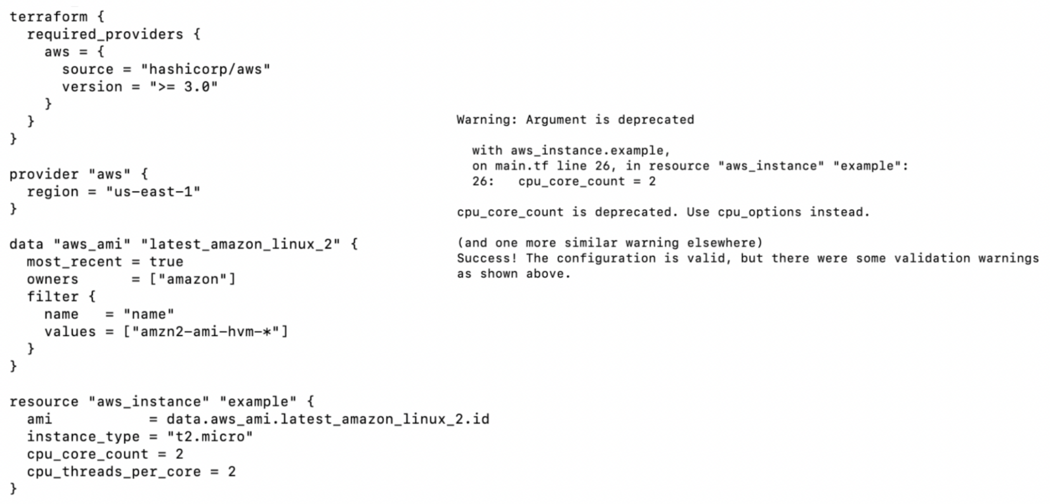
From analyzing the training loss versus the validation metrics, it is clear that the model is training extremely well — however, this is not reflected too much in its final test scores. At first glance, it may seem like the model might be overfitting, but it is very important to notice that our model's score for training was still not much higher than our model's score for validation.



```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = ">= 3.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
}

# ElastiCache user resource with Redis
resource "aws_elasticache_user" "example" {
  user_id              = "example-user"
  engine               = "redis"
  access_string        = "on +noauth"
  passwords            = ["example-password"]
  replication_group_id = aws_elasticache_replication_group.example.id
}

# Example ElastiCache Replication Group for reference
resource "aws_elasticache_replication_group" "example" {
  replication_group_id = "example-group"
  replication_group_description = "Example replication group"
  node_type = "cache.t2.micro"
  engine = "redis"
  port = 6379
  parameter_group_name = "default.redis3.2"
  subnet_group_name = aws_elasticache_subnet_group.example.name
}

# Example ElastiCache Subnet Group for reference
resource "aws_elasticache_subnet_group" "example" {
  name       = "example-subnet-group"
  subnet_ids = ["subnet-12345678", "subnet-87654321"]
}

# IAM Policy Document for ElastiCache user
data "aws_iam_policy_document" "example" {
  statement {
    actions = [
      "elasticache:DescribeUsers",
      "elasticache:ModifyUser",
      "elasticache:DeleteUser",
    ]

    resources = ["*"]
  }
}
```

```
Error: Missing required argument

  on main.tf line 15, in resource "aws_elasticache_user" "example":
  15: resource "aws_elasticache_user" "example" {

The argument "user_name" is required, but no definition was found.

Error: Unsupported argument

  on main.tf line 20, in resource "aws_elasticache_user" "example":
  20:   replication_group_id = aws_elasticache_replication_group.example.id

An argument named "replication_group_id" is not expected here.

Error: Missing required argument

  on main.tf line 24, in resource "aws_elasticache_replication_group" "example":
  24: resource "aws_elasticache_replication_group" "example" {

The argument "description" is required, but no definition was found.

Error: Unsupported argument

  on main.tf line 26, in resource "aws_elasticache_replication_group" "example":
  26:   replication_group_description = "Example replication group"

An argument named "replication_group_description" is not expected here.
```

Figure 3: Example output that does not pass the syntax checker, and the syntax errors output by the checker.



```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = ">= 3.0"
    }
  }
}

provider "aws" {
  region = "us-east-1"
}

data "aws_ami" "latest_amazon_linux_2" {
  most_recent = true
  owners      = ["amazon"]
  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*"]
  }
}

resource "aws_instance" "example" {
  ami           = data.aws_ami.latest_amazon_linux_2.id
  instance_type = "t2.micro"
  cpu_core_count = 2
  cpu_threads_per_core = 2
}
```

```
Warning: Argument is deprecated

  with aws_instance.example,
  on main.tf line 26, in resource "aws_instance" "example":
  26:   cpu_core_count = 2

cpu_core_count is deprecated. Use cpu_options instead.

(and one more similar warning elsewhere)
Success! The configuration is valid, but there were some validation warnings
as shown above.
```

Figure 4: Example output that passes the syntax checker. Notice that there are still a couple lines of warnings, which are counted into our syntax line count.

In the case of DPO, this means that while our model may have assigned a higher probability to generate things more similar to our gold examples (note the slightly higher CodeBLEU in the training results), it did not necessarily translate to reproducing the gold example when prompted, which means that downstream evaluation tests would not necessarily be better. While DPO can lead to improved modeling of user preferences, this does not always translate to better performance on our concrete, task-specific evaluation metrics like syntax validity or CodeBLEU. The model might be learning to mimic preference judgments without necessarily learning the precise behaviors that improve our specific metrics.

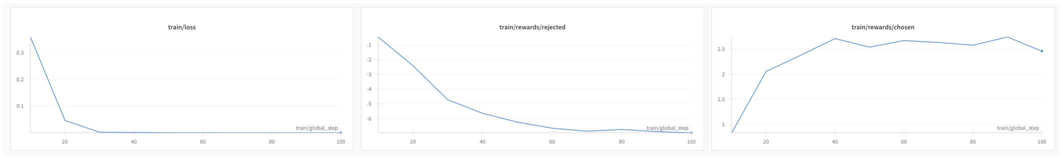


Figure 5: Training loss, rejected reward, and chosen reward. Note that there are 100 training steps, because there are about 320 training examples / 4 gradient accumulation steps / batch size of 4 * 5 epochs = 100 training steps.

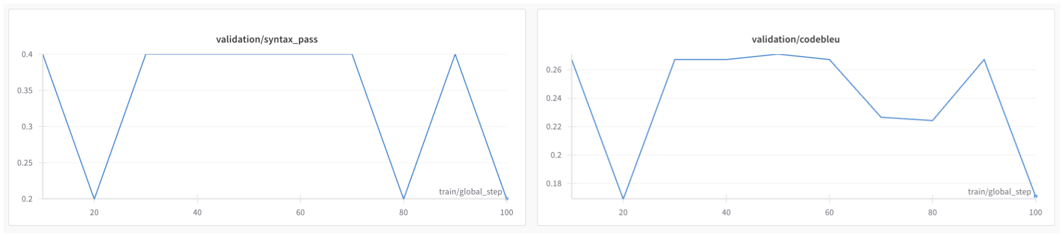


Figure 6: Validation for syntax pass percentage out of 5 examples, and codeBLEU score average for 5 examples.

Another issue may be the limited size of the training dataset relative to the vast output space of IaC code. The IaC-eval dataset contains only a few hundred training examples, which may be insufficient for the model to generalize effectively across the nuanced patterns found in IaC tasks. The lack of data makes it difficult for the model to learn robust correction strategies.

Additionally, the quality of the comparison outputs can introduce challenges. GPT-4o outputs are typically of decently high quality, so when they are used as the "unpreferred" baseline in DPO, the differences between outputs become subtle. This subtlety might have weakened the learning signal that our DPO relied on and could also introduce noise, making it harder for our model to learn meaningful distinctions.

Finally, DPO may not be the best method for the kind of token-level precision required in IaC corrections. IaC syntactical correctness often hinges on small, exact edits, such as modifying indentation, adjusting provider versions, or renaming variables. DPO is a high-level preference-based method, which is not optimized to capture or reinforce such fine-grained, syntax-sensitive changes.

7 Conclusion

Our approach was fundamentally constrained by the limited size of the dataset, which had only about 300 preference pairs, which restricted the diversity of IaC code generation patterns available to the model during training. The output space for IaC corrections is vast and syntactically complex, often requiring token-level precision and structured reasoning. In this context, DPO struggled to internalize the fine-grained edit behavior necessary for robust generalization, especially given the subtlety of differences between outputs and the relatively weak signal from limited preference feedback.

Despite these challenges, our pipeline showed promising signs of potential, particularly in adding a controller layer on top of GPT-4o. Implementing DeepSeek as a corrective layer on top of GPT-4o was immediately effective at improving syntax robustness. Even without fine-tuning, this baseline layer increased the syntax pass rate by 4.01% and cut the average number of syntax errors nearly in half—from 24.89 to 11.72. This demonstrated that large language models like DeepSeek can meaningfully clean up IaC outputs when used in a layered architecture, even with minimal fine-tuning.

After applying DPO fine-tuning to this DeepSeek corrective layer, we obtained further, but very modest, gains in syntactic correctness. Thus, while the model learned to better avoid syntax errors, DPO's relative preference-based objective did not really translate into broader improvements in correctness or task-specific metrics.

Our results suggest that while DPO might help nudge model behavior toward fewer syntactic issues, it is not inherently well-suited to address the kinds of granular, highly structured corrections needed for IaC editing, especially when trained on small datasets with subtle preference distinctions like ours.

For future work, we think that augmenting the training process with synthetic preference data could significantly increase coverage of relevant IaC error patterns and provide much stronger learning signals. Also, integrating external feedback sources, such as our evaluation metrics, as part of a reward model could enable reinforcement learning algorithms better suited to this sort of structured correctness objectives than DPO. Overall, RL combined with controller layer is still promising as a way to improve the value of LLMs for IaC code generation, and perhaps make them consistently usable in industry situations.do

8 Team Contributions

- **Group Member 1: Sophia Zhang** Sophia optimised GPU utilisation, wrote and maintained the training/validation scripts, swept hyper-parameters, ran the full DPO training jobs, and compiled the raw results. She also scheduled and ran the weekly check-ins, tracked action items, handled merge conflicts, and ensured that intermediate artefacts (datasets, LoRA weights, logs) were version-controlled and reproducible.
- **Group Member 2: Aditya Gupta** Aditya designed the controller-pipeline architecture, performed dataset cleaning and prompt–pair generation, implemented the CodeBLEU and Terraform validation metrics, and built the LoRA+DPO fine-tuning harness. He generated GPT-4o baselines, analysed the experimental outputs, produced the figures/tables, and drafted the technical sections of the report and poster.

Changes from Proposal We decided not to use Terraform Rego validation because it took too long to run each time. We also changed our model from Qwen-0.5B to DeepSeek-Coder-1.3B-Instruct because of its better performance for handling code (after testing Qwen and seeing its poor performance on our tasks).

References

Patrick Tser Jern Kon, Jiachen Liu, Yiming Qiu, Weijun Fan, Ting He, Lei Lin, Haoran Zhang, Owen M. Park, George S. Elengikal, Yuxin Kang, Ang Chen, Mosharaf Chowdhury, Myungjin Lee, and Xinyu Wang. 2024. IaC-Eval: A Code Generation Benchmark for Cloud Infrastructure-as-Code Programs. In *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (Eds.), Vol. 37. Curran Associates, Inc., 134488–134506. https://proceedings.neurips.cc/paper_files/paper/2024/file/f26b29298ae8acd94bd7e839688e329b-Paper-Datasets_and_Benchmarks_Track.pdf

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290 [cs.LG] https://arxiv.org/abs/2305.18290