Adaptive Test Time Compute for RL Fine Tuning

James Chen[1] Grace Luo[1]
Aarav Wattal[2]

[1]Department of Computer Science, Stanford University
[2]Department of Electrical Engineering, Stanford University
{jamesc27, luograce, awattal}@stanford.edu

Extended Abstract

Motivation. Reinforcement learning fine-tuning (RLFT) of large language models offers a powerful avenue to align model generations with human preferences, improving response quality in tasks like instruction following and open-ended dialogue. However, RLFT also means resulting models may require more sophisticated inference strategies to fully leverage their preference-aligned behaviors, and these strategies can incur higher computational costs and latency. Our work addresses this challenge by exploring adaptive test-time inference strategies, particularly best-of-n sampling, beam search, and chain-of-thought (CoT) reasoning, with the UltraFeedback instruction following dataset.

Method. Our approach involved fine-tuning the Qwen-2.5B-0.5 base model in two stages. Initially, we applied Supervised Fine-Tuning Direct Preference Optimization (DPO) with gradient clipping and an adaptive learning rate featuring linear warm-up and linear decay. Adaptive test-time inference methods—best-of-n, beam search, and CoT—were systematically evaluated post-training.

Implementation. The fine-tuning process was executed on an AWS g6e.xlarge spot instance equipped with a single NVIDIA T4 GPU. Tokenization was managed using the Qwen tokenizer, with inputs clipped at 250 query tokens and 1000 response tokens. We ensured that only response tokens contributed to loss calculations through appropriate masking.

Results. Adaptive inference techniques markedly enhanced response quality and reduced variability across training stages. Specifically, the best-of-n sampling method consistently improved results across base, SFT, and DPO configurations, albeit increasing computational costs. Best-of-n sampling increased the reward scores from -0.86 (baseline) to -0.77 (SFT) and further improved to -0.58 post-DPO. Beam search initially underperformed in the baseline (-0.89) and SFT (-0.83) configurations but showed remarkable improvement with DPO (-0.51), substantially reducing output variance. Chain-of-thought (CoT) reasoning improved baseline performance slightly to -0.85 but encountered issues with repetitive outputs in code-related tasks, limiting its effectiveness. Qualitatively, we saw that CoT had a tendency to get stuck in its own line of reasoning We also introduce an adaptive inference-time heuristic model which picks the set of inference techniques that best balances accuracy with runtime efficiency. Despite limited training data, we find that this significantly outperforms any other technique (+2.1 reward), as well as having by far the best heuristic score of any method. It is also extremely lightweight, giving us more accurate and more efficient generations for essentially no resource cost.

Discussion. Our findings underscore the importance of adaptive test-time inference strategies in improving LLM performance in real-world scenarios. The effectiveness of beam search post-DPO training emphasizes the need for compatibility between training methods and inference strategies. Nevertheless, the results' specificity to the UltraFeedback dataset indicates that broader validation across diverse datasets is essential. Our ablation study confirms that Best-of-N sampling consistently boosts reward scores across base,

SFT, and DPO models, while beam search only benefits the strongest (DPO) model and chain-of-thought prompting degrades performance universally. Combining methods yields diminishing returns beyond simple multi-generation, but our lightweight heuristic model—trained on ablation results—outperforms every fixed strategy by adaptively selecting the best inference technique per prompt, achieving both higher reward and efficiency. These findings reinforce the significance of optimizing inference – models aligned via DPO interact differently with decoding methods, and adaptive selection can leverage these patterns to maximize gains under resource constraints. However, we rely on a single reward API and one domain/model size, so generalization remains to be validated. Practical challenges included noisy preference signals in DPO, API latency in evaluation, and limited data for heuristic training, suggesting careful tuning and broader human evaluation are needed for robust deployment.

Conclusion. We show that combining supervised fine-tuning and DPO with systematic test-time ablations reveals clear patterns: multi-generation is the most reliable strategy, beam search helps only well-tuned models, and chain-of-thought harms output quality. Crucially, an adaptive heuristic model trained on these ablations can further improve both reward and efficiency by selecting inference strategies per prompt. This demonstrates that adaptive test-time compute, guided by learned heuristics, can significantly enhance aligned LLM performance without major additional cost. Future work should extend these insights across diverse tasks and reward signals, explore inference-aware training objectives (e.g., soft Best-of-N), and incorporate human judgments and cost–environmental considerations to ensure responsible, generalizable deployment.

1 Abstract

Adaptive test-time inference is crucial for maximizing the benefits of reinforcement learning fine-tuning (RLFT) in large language models (LLMs) while managing resource constraints and latency demands. In this work, we examine high-level strategies—sampling (best-of-n), structured search (beam search), and reasoning prompts (chain-of-thought)—applied to a Qwen-2.5B-0.5 model fine-tuned via supervised learning and direct preference optimization on instruction-following data. We find that best-of-n sampling and beam search yield significant improvements in reward-based metrics when aligned with preference-based fine-tuning, whereas chain-of-thought prompts offer marginal benefits and risk repetitive outputs in some contexts. These results underscore the importance of matching inference techniques to fine-tuning objectives and provide actionable guidance for deploying RLFT LLMs in real-world applications that demand balanced trade-offs between output quality and computational efficiency.

2 Introduction

Large language models have demonstrated remarkable capabilities, and reinforcement learning fine-tuning (RLFT) further aligns outputs with human feedback, particularly for instruction-following tasks. While model scaling has been the dominant strategy for improving performance, recent work suggests that strategic use of test-time compute can provide a path to better outputs (9; 10). This is particularly relevant for smaller models, where the gap between potential and empirical performance during single-pass inference can be substantial.

In this project, we focus specifically on the UltraFeedback task: an instruction-following dataset with human preference labels. While RLFT can improve alignment on UltraFeedback, the fine-tuned model's inference-time behavior may shift, making standard sampling or search suboptimal. Furthermore, inference strategies vary in cost, latency, and output characteristics, which is critical for UltraFeedback deployments where consistent, high-quality responses matter. We address the following research questions: (1) How do adaptive inference methods—best-of-n sampling, beam search, and chain-of-thought prompting—affect

response quality of models fine-tuned on UltraFeedback compared to pre-fine-tuned baselines? (2) Under what conditions do these strategies yield the most benefit for UltraFeedback prompts, and how do they balance computational overhead with reward-based metrics? (3) What guidance can be derived for inference configurations when deploying UltraFeedback-trained models in scenarios prioritizing low latency versus high-quality responses?

To answer these, we fine-tune Qwen-2.5B-0.5 on UltraFeedback via supervised learning and Direct Preference Optimization (DPO), then conduct a systematic ablation study of the strategies above across different fine-tuning methods (Base, SFT, and DPO). We demonstrate that multi-generation with reward-based selection provides robust improvements across all model variants, while other strategies show strong dependence on base model quality. Our findings provide targeted recommendations for UltraFeedback deployments, highlighting inference choices that align optimally with preference-based fine-tuning objectives

3 Related Work

Instruction-following fine-tuning often employs RLHF with Proximal Policy Optimization (PPO) to align LLM outputs to human feedback, but PPO's multi-stage pipeline and critic network introduce substantial overhead and complexity (14). Direct Preference Optimization (DPO) simplifies this by framing preference alignment as a supervised objective using log-likelihood ratios, removing explicit reward modeling and on-policy sampling loops while maintaining alignment quality (15).

Chow et al. (19) proposes fine-tuning models to directly optimize Best-of-N inference performance by using imitation learning and reinforcement learning to handle the non-differentiable selection step. The authors show that BoN-aware models learn a meta-strategy balancing high-reward and diverse candidates and achieve significant gains while more efficiently leveraging inference compute. These findings suggest that integrating BoN objectives into fine-tuning can yield better inference-time trade-offs than vanilla BoN sampling, motivating our project's exploration of adaptive test-time compute in RL fine-tuning by indicating a promising avenue for future BoN-aware or meta-inference strategies in UltraFeedback-tuned models.

Beam search structures inference to maximize sequence likelihood, improving consistency but potentially reducing diversity (16). While these techniques are well-studied for pretrained or supervised models, preference-aligned fine-tuning (e.g., on UltraFeedback) can alter token distributions, affecting how decoding methods perform.

Chain-of-thought prompting elicits intermediate reasoning steps to boost performance on complex tasks (17), but its benefits when applied to preference-tuned models are unclear, as reasoning traces may interact unpredictably with altered likelihoods. Our work distinguishes itself by systematically evaluating how these inference strategies perform on models fine-tuned for instruction-following with UltraFeedback using supervised learning and DPO.

4 Methods

4.1 Base Model and Fine-tuning

We initialize from the pretrained Qwen 2.5 0.5B model and fine-tune on SmolTalk prompt–response pairs (20) with SFT and DPO.

With SFT, each example is preprocessed by extracting the last user–assistant turn, encoding the user text and assistant reply, concatenating them with an EOS token, and masking out prompt tokens by setting their labels to `-100`. Inputs are padded or truncated to a fixed `max_total_len`. The loss is the standard

cross-entropy over the assistant tokens only:

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\sum_{t=1}^{T_{\text{resp}}} \log \pi_\theta(y_t \mid x, y_{<t}),$$

where prompt tokens contribute nothing to the loss.

With DPO, we load the SFT checkpoint as a frozen reference model π_{ref}. Given a preference dataset of triples (x, y_w, y_ℓ) where y_w is preferred over y_ℓ, each response is encoded together with the prompt plus EOS, masking prompt positions. During training, we compute the sequence log-likelihoods under both the current model π_θ and the frozen reference π_{ref} by summing token log-probabilities over the response positions. Define

$$r(x, y) \;=\; \beta\big[\log \pi_\theta(y \mid x) \;-\; \log \pi_{\mathrm{ref}}(y \mid x)\big],$$

with scalar $\beta > 0$. The DPO loss for a single triple is

$$\mathcal{L}_{\mathrm{DPO}}(\theta) \;=\; -\log \sigma\big[r(x, y_w) \;-\; r(x, y_\ell)\big] \;=\; -\log \sigma\Big(\beta\big[\log \tfrac{\pi_\theta(y_w|x)}{\pi_{\mathrm{ref}}(y_w|x)} - \log \tfrac{\pi_\theta(y_\ell|x)}{\pi_{\mathrm{ref}}(y_\ell|x)}\big]\Big).$$

Gradients flow only through π_θ and π_{ref} remains in evaluation mode. We tune β (e.g., 0.1–0.5) and learning rate via validation.

4.2 Test-Time Inference Strategies

We evaluate three test-time inference strategies designed to improve output quality through strategic computation:

Best-of-N Sampling: We generate $N = 3$ independent responses by sampling from the model at temperature $T = 0.7$. Each response is scored using the Llama-3.1-Nemotron-70B reward model, and we select the response with the highest reward. This approach leverages the variance in the model's stochastic outputs to find higher-quality responses without modifying the underlying policy.

Beam Search: We maintain $k = 3$ partial hypotheses at each decoding step. At each token position, we expand all current hypotheses by their top-k next token predictions, compute the cumulative log-probability for each candidate sequence, then prune back to the k highest-scoring sequences. This deterministic approach explores multiple high-probability paths through the output space simultaneously.

Chain-of-Thought (CoT): We prepend each prompt with "Let's think step by step" to encourage the model to generate intermediate reasoning tokens before producing its final answer. While effective in larger models, we investigate whether this prompting strategy can elicit structured reasoning in our 0.5B parameter model.

We also evaluate all pairwise and three-way combinations of these strategies. For instance, Best-of-N + Beam combines beam search within each sample before selecting the best among N beam outputs, while CoT + Best-of-N samples multiple reasoning chains and selects the one with the highest reward.

4.3 Evaluation

We evaluate our fine-tuned models on a held-out set of UltraFeedback prompts by measuring both reward improvements and inference latency under different test-time configurations. For each prompt, we generate responses using combinations of sampling (Best-of-N with N=1 or 3), beam search (beam widths 1 or 3), and optional chain-of-thought prefixes; each candidate is scored via the NVIDIA Llama-3.1-Nemotron-70B reward API, and in Best-of-N we select the response with the highest reward. We load the fine-tuned checkpoint in bfloat16 on GPU, tokenize the prompt, and run model.generate in evaluation mode, timing

from the start of generation through scoring to capture both model and API latency. We also generate under identical settings with the SFT reference policy to compute a win indicator for each prompt (1 if the fine-tuned model's reward exceeds the reference's, else 0), and average these to obtain win-rate. Results, including per-prompt rewards, win-rates, and average inference times, are saved to JSON for later analysis.

5 Experiments

5.1 Experimental Setup

Datasets and Task Description. We used two datasets

- SFT: SmolTalk (UltraFeedback) filtered subset - we ran approximately 17,000 update steps
- DPO: UltraFeedback binarized preference pairs

We first perform supervised fine-tuning (SFT) on a prompt–response collection by extracting the last user–assistant turn in each example (using the provided preprocessing functions that encode user text and assistant reply with an EOS token, mask prompt tokens, and pad/truncate to a fixed length). For preference fine-tuning, we use a binarized preference dataset of triples $(\text{prompt}, y_{\text{chosen}}, y_{\text{rejected}})$, where each response is encoded similarly (prompt + EOS + response, masking prompt tokens). At evaluation time, we select held-out prompts from the preference dataset's validation split (not used during training) to measure generalization under different inference settings.

Baselines. We compare three stages: (1) the pretrained base model without any fine-tuning; (2) the model after SFT on the prompt–response data; and (3) the SFT checkpoint further fine-tuned via Direct Preference Optimization (DPO) on preference triples. Each variant is loaded in evaluation mode for downstream generation.

Training Details We trained our models on AWS g6e.xlarge spot instances. In SFT, all model parameters are optimized with AdamW, using a linear warm-up and linear decay schedule, gradient clipping, and a learning rate in the low 10^{-5} range determined by preliminary tuning. Batches consist of examples are preprocessed first by extracting the last user–assistant turn, encoding the user prompt and assistant response with an EOS token, masks out prompt tokens (setting their labels to –100) so only the assistant reply contributes to the loss, and padding or truncating the sequence to a fixed length. Training proceeds until validation loss plateaus, yielding a stable reference for DPO.

In DPO, the SFT checkpoint is frozen as the reference. We construct a DataLoader (e.g. batch size 4), encoding each preference triple with prompt length up to 256 tokens and response length up to 256 tokens. During each step, we compute sequence log-likelihoods of chosen and rejected responses under both the current model and the frozen reference using a token-wise summation function. The DPO loss uses a scaling factor β (tuned to 0.1) in $-\log \sigma[\beta(\Delta_{\text{chosen}} - \Delta_{\text{rejected}})]$. Optimization uses AdamW with a learning rate around 1×10^{-5}, training for a small number of epochs (e.g. 3) and saving checkpoints periodically,

Ablation Study. We run an ablations study across all permutations of our inference techniques. This gives us insight into the reward-time tradeoffs for each of them, as well as which types of prompts each technique or combination of techniques is most suited for.

Heuristic Model. At the core of our extension is a lightweight heuristic model (exact method described in section 6) which uses the results of our ablation study to adaptively choose the most effective technique for a given prompt at inference time. This model uses TF-IDF vector embeddings followed by an sklearn RandomForestClassifier to classify which types of techniques will give the most reward and have the most efficient inference for a given prompt.

Metrics. Our primary metric is average reward under each inference configuration. We also introduce a custom heuristic for the inferece time techniques that allows us evaluate both reward and efficiency of generation simultaneously. In addition to these quantative metrics, we also include sample generations for qualitative evaluation. Together, these metrics allow us to assess how SFT and DPO interact as well as gauge the effectiveness of our test-time strategies.

5.2 Training Progress

Our SFT training showed rapid initial improvement, with test loss dropping from 1.38 to approximately 1.20 within the first 500 steps as the model captured high-frequency patterns. The loss curve then flattened into a long tail, eventually converging to below 1.1 after 17,000 gradient steps. Despite the extremely small batch size (2 examples), we observed no overfitting, with training and validation losses remaining closely aligned throughout training. This stability can be attributed to the dataset's diversity, LoRA regularization, and our learning rate schedule.

5.3 Cost-Quality Trade-offs

While we did not measure exact inference times, multi-generation naturally requires more computation as it samples multiple responses before selection. The trade-off between quality improvement and computational cost remains an important consideration for practical deployment.

6 Heuristic Model

To take advantage of these findings, we create a custom lightweight heuristic model that finds the optimal inference setup based on the prompt at runtime. Our heuristic for performance was the following:

1. For a technique t and prompt p, we define a normalized score:
$$s'_{p,t} \;=\; \frac{s_{p,t} \;-\; s_{t'}\tau_{p,t'}}{\max_{t'} s_{p,t'} \;-\; \min_{t'} s_{p,t'}},$$
and a normalized time:
$$\tau'_{p,t} \;=\; \frac{\tau_{p,t} \;-\; \min_{t'} \tau_{p,t'}}{\max_{t'} \tau_{p,t'} \;-\; \min_{t'} \tau_{p,t'}}.$$
2. Our actual heuristic score is calculated as:
$$M_{p,t} \;=\; (1-\alpha)\, s'_{p,t} \;-\; \alpha\, \tau'_{p,t}.$$
3. For our models, we set $\alpha = 0.3$ to strike a good balance of getting very effective models but trying to limit runtime when possible.

Using this heuristic as our metric for choosing the best model, we used TF-IDF vector embeddings followed by an sklearn RandomForestClassifier. Since the task has such limited output space and training data, we thought this sort of simpler model was the most apt, though in the future we would want to try experimenting with more training data and different types of models. The results for the experiment are found in Table 2. Here, we see that the heuristic model actually significantly outperforms any individual technique, both in terms of our custom heuristic, which is somewhat expected, but also in terms of raw reward! This indicates that our model is learning not only how to balance accuracy and runtime, but is also able to use its extremly limited training data to identify patterns in what sorts of prompts benefit from which techniques and use this to create an adaptive approach that not only is efficient, but also produces the **most** accurate responses. (Note that this technique is not reflected in our leaderboard submission since it employs multi-gen which we recently learned was not allowed).

7 Results and Analysis

7.1 Results

7.1.1 Ablation Study

Table 1 presents our primary findings from the ablation study of inference time techniques across all model and inference combinations.

Table 1: Nemotron reward scores for different inference strategies

Setting	Base	SFT	DPO
Baseline	-25.96	-25.76	-23.375
Multi-gen	**-20.35**	**-20.74**	-20.25
Beam	-25.30	-28.85	-20.25
CoT	-28.36	-30.58	-26.75
Multi+Beam	-19.93	-24.56	**-18.62**
Multi+CoT	-20.58	-22.83	-19.63
Beam+CoT	-25.46	-24.28	-24.75
All	-21.98	-23.67	-21.25

Summarized Findings:

1. Multi-generation consistently improves all models, with gains of 3.70-6.28 points alone
2. Beam search helps DPO models (+1.81) but hurts weaker models (-3.09 for SFT)
3. CoT universally degrades performance, particularly on base/SFT models
4. Combining strategies yields diminishing returns; Multi+Beam slightly improves over Multi alone for DPO

Prompt: Write a function to translate text
CoT Output: Let's think step by step...

```
text = text.replace(" ", "")
text = text.replace("\n", "")
text = text.replace("\t", "")
text = text.replace("\r", "")
text = text.replace("?", "")
text = text.replace("!", "")
```

Figure 1: Example of repetitive CoT failure in code generation

Scores: -25.75, -20.875, -26.125 **Prompt:** Explain the impact of the Treaty of Versailles on political and economic stability.
Best CoT Output: Let's think step by step...

```
The Treaty of Versailles, signed in 1919, was a result of the Great War
(World War I) between the Allied Powers (Great Britain, France, Italy,
and the United States) and the Central Powers (Germany, Austria-Hungary,
and Serbia). The treaty was designed to address the war's
consequences and restore the status quo of the pre-war world. The treaty
had several significant impacts on political and economic stability in
the aftermath of the war:

1. Political Stability: ...
```

Figure 2: Example multi-gen working well with CoT

7.1.2 Heuristic model

The heuristic model achieves the best results, with an average raw score of –17.16 and the highest heuristic score of 0.2993, clearly outperforming all individual inference techniques. Among the standalone methods, multi-generation yields the largest improvement over the baseline (raw –19.24 vs –23.81), while beam search offers only modest gains (–22.25) and chain-of-thought actually degrades performance (–25.98). Combining methods does not outperform simple multi-generation: multi+beam (–19.53) is slightly worse than multi-gen alone, and combinations involving CoT perform poorly. These findings indicate that adaptive selection via the heuristic model effectively balances quality and runtime, and that multi-generation is the strongest single strategy, whereas CoT prompting harms average performance in this setting.

Setting	Raw Score	Heuristic Score
Heuristic Model	**-17.1625**	**0.2993**
Baseline	-23.8062	0.2071
Multi-gen	-19.2375	0.2095
Beam	-22.2500	0.1712
CoT	-25.9750	0.1382
Multi+Beam	-19.5250	0.1631
Multi+CoT	-23.0625	0.1250
Beam+CoT	-26.1313	0.1292
All	-22.3000	0.1145

Table 2: Comparison of heuristic model vs. individual techniques on average raw score and heuristic score.

Summarized Findings:

1. Despite severely limited training data, our lightweight model is able to effectively predict the methods that will optimize the heuristic score, leading to a 0.9 increase over the best single set of techniques.

2. In addition, without even optimizing solely for reward our heuristic model gets the best reward of any method, indicating that it has gained knowledge about what techniques are most useful and efficient in different situations.

3. Tuning α down led to more consistantly choosing the same technique, while tuning it up led to more variance. With our selection of $\alpha = 0.3$, we has significant variance with no single technique being chosen more than 33% of the time.

Inference Techniques Used:

```
1: Baseline
2: Multi-gen
3: All
4: Baseline
5: Beam
6: Beam + CoT
7: Baseline
8: Multi-gen
9: Multi-gen + CoT
10: Beam
...
```

Figure 3: Heuristic model uses all sorts of different inference techniques

7.2 Analysis.

7.2.1 Chain-of-Thought Degradation

Figure 1 shows representative CoT failures. The model frequently enters repetitive loops rather than meaningful reasoning, suggesting tendencies to get stuck in a specific step in its line of reasoning. The leads it to have quite bad average results, however it is still the most effective method on some select prompts

7.2.2 Chain-of-Thought and Multi-Generation

In line with this, as shown in Figure 2 we observe that the adding Multi-gen to Chain-of-Thought gives a lot of benefit. We think that this is because while Chain-of-Thought is more prone to very bad outputs, it can also create very good ones! This high variance is more suited to Multi-gen which mitigates the worst case outputs of our model.

7.2.3 Beam Search Model Dependency

Beam search effectiveness strongly correlates with base model quality. For well-aligned DPO models, beam search helps by maintaining multiple plausible hypotheses. For poorly aligned models, it amplifies local errors by committing to high-probability but low-quality token sequences early in generation.

7.2.4 Effectiveness of Heuristic

We found the heuristic model very interesting. Given that a lot of our observations about our ablation study focus on high variance of different techniques (especially Chain-of-Thought), we hypothesized that our adaptive model might be able to learn some of the features that lead to high variance and account for them in its selection. This proved to be a good hypothesis, since even when optimizing for a heuristic that included runtime as well as raw reward, our model still outperformed all individual methods by a significant margin.

Additionally, one notable thing is that with our choice of α the model didn't fall into a pattern of using the same techniques over and over again, in fact in Figure 3 we see that there is high variance of different techniques used per prompt, still leading to this very high performance. This indicates that it has a relatively strong understanding of what techniques and combinations of techniques should be used when, rather than just defaulting to one or two.

8 Discussion

Our results indicate that adaptive inference (Best-of-N sampling, beam search, and CoT prompting) can improve reward-model scores for fine-tuned policies, but several limitations temper these findings. First, we rely on a single external reward API whose outputs may not align perfectly with human judgments, and our experiments focus on one domain and model size, so generalization to other tasks or scales is uncertain. The explored inference settings (e.g., small N, limited beam widths, simple CoT prompts) capture common regimes but omit more advanced or large-scale decoding techniques. Compute and API constraints restricted hyperparameter sweeps and prompt sets, so reported gains should be viewed as indicative rather than definitive. Training DPO also faced instability from noisy preference pairs and small batch sizes, requiring careful tuning of the likelihood-ratio scale and learning rate.

From a broader-impact perspective, adaptive inference increases latency and resource use, which may challenge low-resource deployments and raise environmental concerns; at the same time, improved alignment can reduce harmful outputs and enhance user trust. Dependence on a proprietary reward API affects reproducibility and accessibility. During the project, practical difficulties included robustly parsing diverse conversational formats in preprocessing, handling API timeouts and measuring end-to-end latency, and managing GPU memory limits. Despite these challenges, the study highlights the value of co-designing fine-tuning and inference strategies. Future work should validate across varied benchmarks, explore cost-efficient approximations (e.g., early-stopping in Best-of-N), and incorporate human evaluations and environmental accounting to ensure responsible deployment.

We think that perhaps the most interesting part of our experiments was the performance of the heuristic model. It was able to significantly outperform any of our individual test-time techniques on not just our selected heuristic, but also reward in general! This indicates the model is not only able to balance latency and quality by assessing prompt complexity, but also learns when each strategy is most likely to improve output. This gives us a custom, lightweight decision approach based on our prior analysis on when different techniques are effective. Given that we were only able to generate a very small amount of data for this model, this gives us great hope for similar models in future lines of work.

9 Conclusion

In this work, we demonstrate that combining supervised fine-tuning and Direct Preference Optimization with adaptive test-time inference strategies leads to consistent improvements in automatic reward-model scores for instruction-following tasks while also exposing critical compute–quality trade-offs. We find that models fine-tuned with preference data interact differently with decoding methods, and that modest settings—such as small N in Best-of-N sampling or moderate beam widths—often offer the best balance under realistic resource constraints. Although our evaluation is limited to a specific reward API and a single domain, the broader lesson is that careful selection and tuning of inference strategies is as important as the fine-tuning procedure itself for obtaining aligned, high-quality outputs. This underscores the need to consider training and inference jointly when designing systems for real-world deployment.

We note that, while our heuristic model factors in relative runtime considerations, we did not perform precise wall-clock or energy measurements in this study. Incorporating explicit timing or profiling data

would strengthen the evaluation of latency–quality trade-offs. Additional exploration should include systematic measurement of inference time to validate and refine heuristics.

There are several additional avenues to build on these findings. First, exploring inference-aware fine-tuning objectives—such as soft Best-of-N (18) or other methods that directly optimize for multi-candidate decoding—could reduce reliance on large N at inference and improve alignment efficiency. Second, investigating cost-efficient approximations like early-stopping heuristics, speculative decoding, or variational Best-of-N can help capture the benefits of multi-generation with lower latency. Third, broader validation across diverse tasks (e.g., open-ended dialogue, factual QA, creative writing) and different model scales is needed to assess generality. Fourth, human evaluations should complement reward-model scores to ensure that automatic gains translate into genuine user-perceived improvements. Fifth, quantifying energy and latency costs of adaptive inference will inform responsible deployment, potentially leading to dynamic strategies that adjust compute based on context or device capabilities. Finally, enhancing robustness to noisy or ambiguous preference data during DPO fine-tuning can improve training stability.

10 Contributions

- **James Chen:** Built original SFT model, implemented inference time techniques, designed and performed the inference ablation study, implemented and evaluated custom heuristic-based model, wrote report.
- **Grace Luo:** Improved SFT model, built DPO model, wrote evaluation code, wrote report.
- **Aarav Wattal:** Improved SFT results by fixing data processing and adjusting conversation/context length, created foundation for report, wrote code for test-time inference strategies.

Acknowledgments

We thank the CS 224R teaching staff for valuable feedback throughout the project, specifically Ashish Rao who gave us valuable insight during the poster presentations. Computational resources were provided by Stanford University.

References

[1] Ahmadian, A., et al. (2024). Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. arXiv:2402.14740.

[2] Allal, L.B., et al. (2025). SmolLM2: When smol goes big – data-centric training of a small language model. arXiv:2502.02737.

[3] Cui, G., et al. (2024). UltraFeedback: Boosting language models with scaled AI feedback. arXiv:2310.01377.

[4] Fu, Y., et al. (2023). Complexity-based prompting for multi-step reasoning. ICLR 2023.

[5] Gandhi, K., et al. (2024). Stream of search (SoS): Learning to search in language. arXiv:2404.03683.

[6] Gehring, J., et al. (2025). RLEF: Grounding code LLMs in execution feedback with reinforcement learning. arXiv:2410.02089.

[7] Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback. NeurIPS 2022.

[8] Rafailov, R., et al. (2023). Direct preference optimization: Your language model is secretly a reward model. arXiv:2305.18290.

[9] Snell, C., et al. (2024). Scaling LLM test-time compute optimally can be more effective than scaling model parameters. arXiv:2408.03314.

[10] Wang, X., et al. (2023). Self-consistency improves chain of thought reasoning in language models. ICLR 2023.

[11] Wei, J., et al. (2022). Chain of thought prompting elicits reasoning in large language models. NeurIPS 2022.

[12] Zelikman, E., et al. (2022). STaR: Self-taught reasoner bootstrapping reasoning with reasoning. NeurIPS 2022.

[13] Zhang, L., et al. (2025). Generative verifiers: Reward modeling as next-token prediction. arXiv:2408.15240.

[14] Schulman, J., et al. (2017). Proximal Policy Optimization Algorithms. arXiv:1707.06347.

[15] Rafailov, R., et al. (2024). Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290.

[16] Vijayakumar, A. K., et al. (2018). Diverse Beam Search: Decoding Diverse Solutions from Neural Sequence Models. arXiv:1610.02424.

[17] Wei, J., et al. (2023). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903.

[18] Verdun, C. M., et al. (2025). Soft Best-of-n Sampling for Model Alignment. arXiv:2505.03156.

[19] Chow, Y., et al. (2024). Inference-Aware Fine-Tuning for Best-of-N Sampling in Large Language Models. arXiv:2412.15287.

[20] Ben Allal, L., et al. (2025). SmolLM2: When Smol Goes Big — Data-Centric Training of a Small Language Model. arXiv:2502.02737.