# SpeculativeReasoner: Learning To Offload Reasoning

**Yash Akhauri** **Anthony Fei** **Chi-Chih Chang**
**Ahmed F. AbouElhamayed** **Yueying Li** **Mohamed S. Abdelfattah**
Cornell University, Ithaca, NY
`{ya255, ayf7, cc2869, afa55, yl3469, mohamed}@cornell.edu`

## Abstract

Reasoning with large language models (LLMs) is costly because they often need to generate thousands of tokens before arriving at an answer. Each new token references all prior tokens via Key-Value (KV) Caches, making decoding a memory-bound operations that grows quadratically in latency. However, not all tokens are equally difficult to produce. Some parts of the reasoning process are more complex than others. In this paper, we annotated challenging portions of reasoning traces in over 18K chain-of-thoughts (CoT) from the OpenR1-Math-220k dataset. We use supervised fine-tuning (SFT) and GRPO on a 1.5B-parameter reasoning model, so that it can *learn to offload* difficult parts of its own reasoning process to a larger model. This approach boosts AIME24 accuracy by 25%, while offloading less than 1.5% of the generated tokens to a larger model. We open-source our SpeculativeReasoner Model, Data-set, Code and Logs.

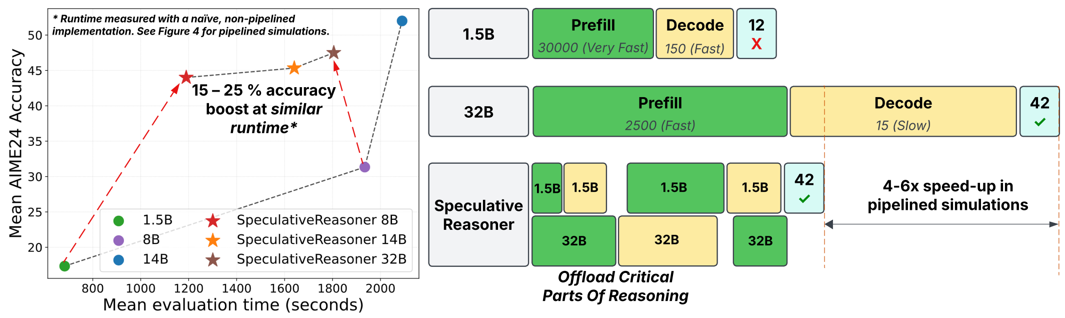


Figure 1: SpeculativeReasoner *controls* the decode process, by intelligently offloading generation to the big model during difficult parts of the reasoning process. Numbers in gray indicate **tokens processed by the model per second** in prefill and decode stages. Leveraging the 1.5B model for majority of the decode process leads to significant end-to-end speedup at inference while improving accuracy over the small model.

## 1 Introduction

Large language models (LLMs) are powerful general-purpose learners that excel at a wide range of tasks [1–3]. Recent advances in LLM post-training have shown that their performance on reasoning-heavy tasks can be improved by inducing the *ability to reason* by generating explicit chain-of-thoughts (CoT) about a question before arriving at the final answer [4]. However, this shift towards more complex, multi-step reasoning during inference [5] significantly increases test-time compute cost. In practice, LLMs often have to generate thousands of tokens while referencing all previously produced tokens (via Key-Value Caches) for every new token. This process is *memory-bound* and grows *quadratically* with respect to sequence length [6–8], making it very time-consuming as we scale up model sizes and rely on longer CoT to improve reasoning [9, 10]

There have been several efforts to further increase compute at test time to improve accuracy on reasoning tasks such as AIME24 and MATH500 [11]. This leads to an explosion in the thinking time needed: thousands of tokens must be decoded before generating the final answer. However, *thinking time* is generally measured in terms of tokens. As test-time compute continues to scale, merely counting tokens becomes a poor measure of true latency, since longer sequences of KV-Cache must be accessed at every decoding step. This cost is not linear, especially during decoding.

A key approach that focuses on this problem with an aim to offer *lossless* speed-up in inference is Speculative Decoding [12]. In this technique, a smaller draft model proposes multiple tokens at once (decoding), and a larger model quickly verifies (via parallelizable prefill) whether the proposed tokens match its own predictions. If they do, these tokens are accepted, otherwise they are rejected and re-decoded. Their key observation is two-fold; **(1)** hard language-modeling tasks include easier sub-tasks that can be approximated by smaller models. **(2)** decode is memory-bound, pre-fill is *less memory-bound* on existing hardware. By combining these insights, speculative decoding has the larger model do *prefill evaluation* (Faster than full-decoding in the large model) of tokens that are actually decoded by the samll model (faster still, relative to the large model). This yields favorable scaling at decode time: the smaller model performs most of the heavy lifting, and larger model needs to quickly validate the generated tokens. [13–16]

While speculative decoding guarantees correctness, this can lead to a very high *rejection rate* from the larger model, wasting several prefills on the large model as well as decode segments from the smaller (draft) model. We posit that such token-by-token verification is *not necessary for reasoning*, and can instead be taught to the draft model. In other words, the draft model itself can learn to identify segments of the reasoning process where it may make a mistake, and simply offload only those segments to the larger model for decoding. Our contributions are:

- We develop and open-source a simple fine-tuning dataset and recipe, to enable models to learn when to offload their own decoding process to a larger model.
- We demonstrate that accuracy of small reasoning models can be improved by 30% by offloading less than 5% of the reasoning process to larger models.
- We show that models can learn when a task is difficult, and can leverage reinforcement learning for optimizing latency (**RL4L**). This enables a new paradigm in which models are taught to align not just with human preferences, but with hardware preferences too.

## 2 Background

**Test-Time Scaling:** Early work on prompting showed that pretrained LLMs can reason if provided explicit CoT traces in the prompt [10]. However, this method is brittle and has a large inference-time token budget requirement. A more robust method to induce reasoning is with **Supervised Fine-Tuning** (SFT) on gold CoTs. The model is shown questions formatted with `<think>` CoT `</think>` answer, which teaches the model to imitate the reasoning trajectory. SFT has been used to *induce* [17] an internal `<think>` stage that can be exploited at test time. However, SFT is fundamentally an imitation procedure, where the policy is rewarded for matching every token in the CoT, even if they are not decisive for getting the right answer. As a consequence, the model doesn't receive a signal to indicate that a particular step is a dead end. Reinforcement Learning (RL) tries to fill this gap, by giving rewards dependent on the *outcome* (correctness) as well as rewards for formatting (for e.g., whether `<think>` tokens were used, answer returned in expected format etc.). Simple outcome-level RLHF only look at final answer, but process-level methods [18–21] also attach reward to intermediate steps. DeepSeek-R1 introduced **Group Relative Policy Optimization (GRPO)**, a lightweight policy-gradient variant that estimates the baseline by z-scoring rewards *within* each sampled group of trajectories, eliminating the value network and halving memory cost[22, 23]. In combination, SFT *induces* the `<think>` (reasoning) behavior; GRPO (and related RL variants) refine it. This two-stage recipe has given rise to several reasoning models, and motivates our own investigation in inducing tokens that can improve accuracy *and* performance.

**Performance Implications Of Test-Time Scaling:** Inference-time reasoning scaling strategies broadly focus on sequential and parallel scaling. Sequential approaches allocate extra compute on a

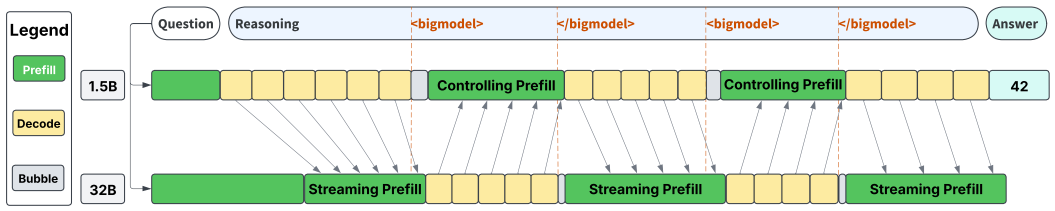


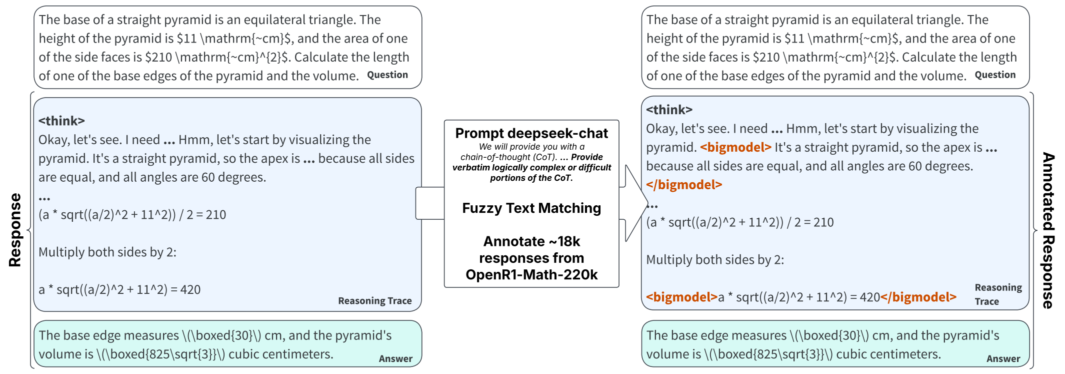
Figure 2: With SpeculativeReasoner, the small model (1.5B) acts as the *controller*. While the small model is decoding, the big model *keeps up* with the generations by doing **streaming prefills** to keep its KV-Cache updated. Once the small model emits `<bigmodel>` tag, the big model takes over generation. At this time, the small model does **controlling prefills**, this serves a dual purpose, keeping the KV-Cache updated, as well as checking if the small model wants to take back control. The generation is halted for the big model if the small model emits `</bigmodel>` during its controlling prefill, and the small model takes over decode.

single chain-of-thought, for e.g., by prompting the model to *think longer* or iterative refine its own output [24]. Such self-refinement allows LLMs to critique and improve its answer, yielding higher accuracy. Parallel approaches run multiple reasoning chains concurrently and aggregate the results, for e.g., using self-consistency, best-of-N voting [9]. Sequential scaling often yields a better return on 'net tokens produced' than parallel [25, 26]. However, these gains come at a significant cost; longer output means more tokens have to be decoded at inference time. Autoregressive generation has two distinct phases – a **prefill** (process input) and **decode** (generate tokens one-by-one). The prefill is a one-time, highly parallel pass over the input sequence. It has large matrix-multiplications that fully utilize the hardware's compute throughput. On the other hand, decode emits tokens one at a time; each step performing small matrix-vector operations and repeatedly fetching key-value caches for *all previous tokens.* [8]. This makes decoding **memory bandwidth bound**, and much slower per token. The decode stage runs at a fraction of peak throughput. Pushing an LLM sequentially to produce very long chain-of-thought incurs quadratic time complexity in sequence length, which is fundamentally more expensive than parallel scaling. Speculative Decoding aims to mitigate decode overhead by using a small draft model to produce a batch of future tokens, and have a large model verify the draft by consuming these tokens in a forward pass (fast prefill) instead of generating them one-by-one. The draft generation can be done concurrently while the large model is busy verifying the previous chunk, which can hide latency. However, it still relies on the large model to check every generation chunk, and this can quickly become a bottleneck if the draft model frequently proposes tokens that get rejected. An alternative path to efficient test-time scaling is to make the small *draft* model itself judge hen to invoke expensive reasoning. In other words, rather than always expending large-model compute to correct potentially mis-matching tokens, relax the exact-match constraint and teach the small model to simple invoke large model decode at crucial parts of the reasoning.

## 3 SpeculativeReasoner

We extend the usual reasoning delimiter `<think>...</think>` with new control tokens `<bigmodel>...</bigmodel>`. These control tokens indicate the *start and end of the offload* to the big model respectively. From Figure 2, the inference flow follows:

- The small model is decoding. At this time, the big model does **streaming prefills**, taking chunks of small model generations and keeping its Key-Value cache updated.
- The small model emits `<bigmodel>`, this suspends the small-model decode, and the big model starts decoding. There is a negligible delay in decode because the big model was doing streaming prefills, and is up-to-date with the current CoT trace.
- While the big model is generating, the small model does **controlling prefills**, taking chunks of big model generations and updating its Key-Value cache, but at the same time, checking its own next-word predictions to check if it emits `</bigmodel>`, which would *take back control* from the big model.
- Once small model emits `</bigmodel>`, the big model halts and switches to **streaming prefill**, as the small model continues the decode.


Response
Prompt deepseek-chat
We will provide you with a chain-of-thought (CoT). ... Provide verbatim logically complex or difficult portions of the CoT.
Fuzzy Text Matching
Annotate ~18k responses from OpenR1-Math-220k
Annotated Response

The base of a straight pyramid is an equilateral triangle. The height of the pyramid is $11 \mathrm{~cm}$, and the area of one of the side faces is $210 \mathrm{~cm}^{2}$. Calculate the length of one of the base edges of the pyramid and the volume. Question

<think>
Okay, let's see. I need ... Hmm, let's start by visualizing the pyramid. It's a straight pyramid, so the apex is ... because all sides are equal, and all angles are 60 degrees.
...
(a * sqrt((a/2)^2 + 11^2)) / 2 = 210

Multiply both sides by 2:

a * sqrt((a/2)^2 + 11^2) = 420 Reasoning Trace

The base edge measures \(\boxed{30}\) cm, and the pyramid's volume is \(\boxed{825\sqrt{3}}\) cubic centimeters. Answer

The base of a straight pyramid is an equilateral triangle. The height of the pyramid is $11 \mathrm{~cm}$, and the area of one of the side faces is $210 \mathrm{~cm}^{2}$. Calculate the length of one of the base edges of the pyramid and the volume. Question

<think>
Okay, let's see. I need ... Hmm, let's start by visualizing the pyramid. <bigmodel> It's a straight pyramid, so the apex is ... because all sides are equal, and all angles are 60 degrees.
</bigmodel>
...
(a * sqrt((a/2)^2 + 11^2)) / 2 = 210

Multiply both sides by 2:

<bigmodel>a * sqrt((a/2)^2 + 11^2) = 420</bigmodel> Reasoning Trace

The base edge measures \(\boxed{30}\) cm, and the pyramid's volume is \(\boxed{825\sqrt{3}}\) cubic centimeters. Answer

Figure 3: We take the entire response for a question from OpenR1-Math-220k and prompt deepseek-chat to annotate difficult portions of the response. These spans are encased in our (`<bigmodel>`, `</bigmodel>`) tags.

In this flow, no modification to the large model is required. The *controlling prefill* mode continuously checks whether to halt the big model generation. Note that the prefill is highly parallel and cheap, so the small model can quickly evaluate when to halt. This method keeps the KV-Cache up-to-date on both models, and either model can resume decoding without delays. Decode is memory-bound, prefill is more compute-bound and cheap, this scheme builds upon the speculative decoding insight on *prefill once, decode when necessary*, but without requiring token-by-token verification. Most of the CoT is entirely produced by the small model.

### 3.1 Training Procedure

Inducing reliable offload boundaries from scratch is tricky: (`<bigmodel> </bigmodel>`) never appear in ordinary text, so there is no incentive to emit them. To address this, we follow a simple two-stage training pipeline.

**Supervised Fine-Tuning:** We sample 18K CoT traces from the Open-R1-Math-220k corpus. For each trace, we prompt deepseek-chat to annotate the most difficult spans. We then do fuzzy-text matching to identify boundaries and wrap these spans with the new control tokens (`<bigmodel>`, `</bigmodel>`). We take these annotations and fine-tune the small model on this corpus to induce the control tokens.

**GRPO refinement:** Supervised traces ensure the tokens appear, but they do not guarantee formatting or rewards for a target offload ratio (to control how much of the decode is offloaded, as it directly impacts latency). We therefore run GRPO on the model post-SFT using a sub-set of the SFT dataset. The rewards combine correctness, formatting and **latency alignment** – a reward for adhering to the desired offload budget (e.g., 10% of the CoT). During GRPO, we do not involve the big model for completions. This means that the primary focus of the reward is on latency, not on accuracy.

This two stage pipeline is cheap, as it does not require the big model to be fine-tuned, and does not involve big-model invocations in the GRPO procedure. Further, `<bigmodel>` can then be offloaded to any larger model, whether its 7B, 14B, 32B etc. This formulation is weakly *latency-aware*, as our offloading reward is directly calculated by simulations on expected speedup. This fine-tuning process can be further improved by real-time latency feedback and accuracy modeling with true offloading.

## 4 Experiments

To create our dataset, we prompt deepseek-chat to annotate the first 18K *generations* from the OpenR1-Math-220k [27] dataset. Our prompt explicitly asks for the 20% most logically complex or difficult portions of the CoT as snippets. We then do fuzzy text matching to ensure the text is identified correctly, and wrap that in the `<bigmodel>...</bigmodel>` tags.

We adopt a two-stage training pipeline to induce offloading behavior to our model. We use `DeepSeek-R1-Distill-Qwen-1.5B` for our small model. We first do supervised fine-tuning of this

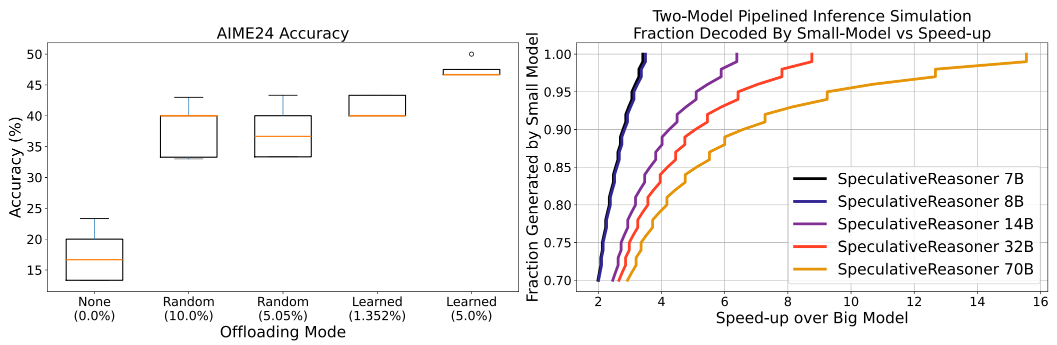


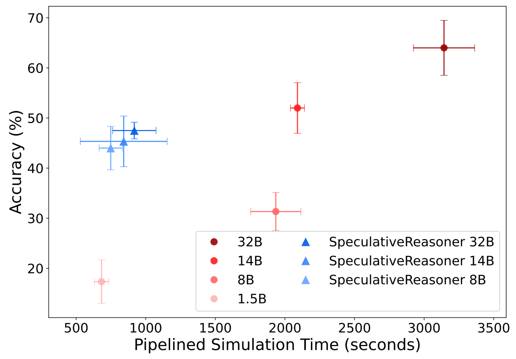
Figure 4: **(Left)** Randomly offloading sections of the decode process from a 1.5B model to 32B model boosts AIME24 accuracy by upto 20%, but if we *learn to offload* correctly, we can achieve higher gains in accuracy with just a 1.35% offload – over $7\times$ fewer offloads than random offloading at 10%. A 5% learned offload enhances performance significantly. **(Right)** We run pipelined performance simulations by profiling a range of models on A6000 GPUs and find that at a 1.35% offload, we can expect 8-9$\times$ faster inference over the big model.

model on $8 \times A6000$ GPUs with a batch size of 64, learning rate of $5e-5$ and warmup ratio is set to 0.05 and is trained for 3 epochs. The learning rate follows a cosine decay schedule to zero. Following Open-R1 [27], we pack all SFT samples to the max sequence length of 16384. The packed samples retain their original positional embeddings. We take the resulting model, and do GRPO fine-tuning on that. We use 14 generations with a batch size of 128, maximum completion length of 4096 with an initial learning rate of $1e-6$. The temperature is set to 0.7, warmup ratio is 0.1 and we follow a cosine decay schedule. We GRPO on only a subset of our dataset (500 random samples from 18K). We primarily rely on `DeepSeek-R1-Distill-Qwen-32B` as our big model, however, since both our SFT and GRPO training formulation do not require involvement from the big model, it is possible to use *any model* as the big model.

For the GRPO procedure, we define a combined reward by summing three components, each weighted equally, to promote correctness, proper formatting and adherence to our desired offloading behavior. First, an *accuracy reward* measures whether the final answer matches the ground truth; if there is a match, theres a +1 reward, else 0. Second, a *format reward* checks whether the entire response follows the `<think>`–`</think>` and `<answer>`–`</answer>` scaffold, awarding +1 for correct scaffolding plus an additional +1 if any `<bigmodel>` tags are properly nested and closed. Finally, a *tag count reward* grants partial credits for the presence of each essential tag (e.g., `<think>` and `</think>`), incentivizes well-formed `<bigmodel>` usage, and includes a coverage-based term that encourages moderate offloading. This coverage term is computed by measuring the fraction of tokens enclosed in `<bigmodel>` blocks and mapping it through a piecewise linear function that increases from 0 to +1 when coverage is below 0.4, then linearly decreases from +1 down to -1 as coverage approaches 1.0. Hence, minimal or excessive offloading is disfavored, while balanced usage is encouraged. If there is a mismatch in the number of `<bigmodel>` opening and closing tags, the reward is penalized, reflecting improper offload boundaries. These three partial rewards—accuracy, format, and tag count—are combined with equal weight into the final scalar reward for each sampled trajectory. Note that our GRPO procedure sets target offload at 0.4 (40%), because we can always reject a `<bigmodel>` request (random-rejection), but we cannot *induce* higher offloading post-finetuning. If we choose to do no offloading, we find that this SFT+GRPO procedure has no noticeable impact in the AIME24 accuracy, so we can always reject `<bigmodel>` request by trading off accuracy up-to the models original baseline accuracy.

## 4.1 Offloading Behavior

We present three primary observations, using a 1.5B model as our small model, and 32B model as our big model. **(1)** Even with random offloading at 5-10%, it is possible to see notable accuracy gains. **(2)** With smart offloading of just 1.35% (median), we are able to see accuracy gains similar to random offloading at 5-10%, indicating that *learning to offload* at critical portions can be very effective. **(3)**



| Model | Acc. (%) | Non-Pipe. (s) | Pipe. (s) |
|---|---|---|---|
| 1.5B | 17.3 | – | 683 |
| 8B | 31.3 | – | 1934 |
| 14B | 52.0 | – | 2090 |
| 32B | 64.0 | – | 3143 |
| SpecR 8B | 44.0 | 1190 | 749 |
| SpecR 14B | 45.3 | 1640 | 842 |
| SpecR 32B | 47.5 | 1805 | 918 |

Figure 5: **(Left)** SpeculativeReasoner can benefit greatly from offloading even to smaller models: SpecR-8B performs almost as well as SpecR-14B and SpecR-32B while offloading only $\sim$5 % of the decode. **(Right)** SpeculativeReasoner Pipe. (Pipelined) evaluation times are simulated by accounting for the 5.54 % offload overhead relative to the 1.5B baseline. Note that our non-pipelined implementations are already faster, but they do use more GPUs for the small + big model. Models used are listed in Section 4.2

With a median learned offloading of 5%, we are able to boost accuracy by upto 30%. We present our results on the AIME24 dataset.

In Figure 4, we compare the AIME24 accuracy of the `DeepSeek-R1-Distill-Qwen-1.5B` model with no offloading, random offloading and learned offloading. Our learned offloading model was trained with SFT + GRPO on 5k samples for this experiment. Even offloading 5-10% of the decode process to the large model is sufficient to boost the AIME24 accuracy by over 20%, however, if this process is *learned* (with SFT + GRPO), we can further improve the accuracy by a few points, while offloading only 1.35% of the decode process to the `DeepSeek-R1-Distill-Qwen-32B` model

## 4.2 Offloading Across Model Sizes

One of the key advantages of SpeculativeReasoner is that only the smallest model needs to *learn to offload*, and our GRPO fine-tuning procedure *does not involve* the larger model, as we rely on SFT to induce `<bigmodel>` tags in difficult regions, and GRPO to simply adhere to reward formatting. This means that we can use *any larger model* as the big model. To study the impact on accuracy across different *big models*, we use `DeepSeek-R1-Distill-Qwen-1.5B` as the small m odel with `DeepSeek-R1-Distill-Llama-8B`, `DeepSeek-R1-Distill-Qwen-14B` and `DeepSeek-R1-Distill-Qwen-32B` as the big model. From Figure 5, we can see that even offloading to the 8B model (SpeculativeReasoner 8B) boosts accuracy significantly. Surprisingly, SpecR 8B consistently performs better than even the base 8B model.

## 4.3 Dataset distribution and inducing offloading

In Figure 6, we analyze our annotated dataset of 18500 reasoning traces. Firstly, we investigate *where* deepseek-chat decides to offload. Specifically, we track the relative positions of `<bigmodel>` spans across all examples, and find that there is a slightly higher bias towards offloading earlier parts of the reasoning process. This is intuitive, as the later parts of the reasoning process may just be inferring to prior *more difficult* reasoning steps from the start. Our data generation procedure also adheres to the 20% offloading target, with majority of the examples offloading less than 20% of the trace.

In Figure 7, we randomly sample 10 questions and graph the spans where the offloads happened. The *high signal* indicates that the span is encased in `<bigmodel>` tag. We find that supervised fine-tuning is not sufficient, as several generations do not have proper offloading behavior. However, after the GRPO procedure, the model is able to offload effectively, following the formatting and frequency requirements. While our GRPO procedure maximizes reward for a offload of 40%, we still empirically observed approximately a 5% offload rate, indicating that our GRPO procedure may need further tuning.

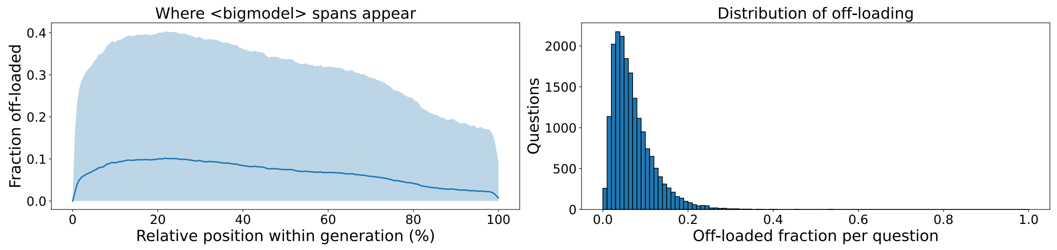


Figure 6: Analysis of our annotated dataset reveals that **(Left)** `<bigmodel>` tags appear relatively uniformly over the text, with slight preference in the earlier part of the reasoning trace and **(Right)** most questions are with-in our desired $< 20\%$ offloading range.

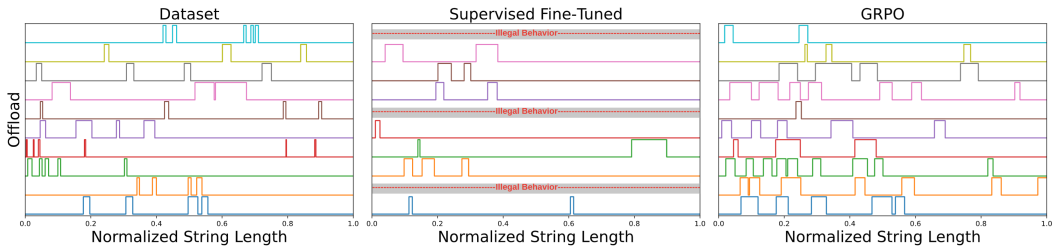


Figure 7: Stacked random samples of offloading behavior from our dataset, the Supervised Fine-Tuned model and the final model post-GRPO. The *high* signal means that part of the decode was offloaded to the big model. *Illegal* offloading behavior indicates that the small model did not take back control or had incorrect formatting. The supervised fine-tuned model offloads less than 1% of the decode and is not reliable, whereas the final model is able to reliably offload decode, adhering to our reward function.

## 4.4 Performance Modeling

For our accuracy evaluations on AIME24, we adapt the lm-evaluation-harness [28] changes from s1 [26] with-in our own framework which uses vLLM for fully-parallel evaluation of all questions efficiently. Our current implementation does not do full-pipelined streaming and controlling prefills. Our accuracy evaluation code does prefill on the big model *after* the small model generates the `<bigmodel>` token. We generate 64 tokens from the big-model at a time, and then the small model decodes 8 tokens *from every running next-token prediction task* (64 completion tasks 8 tokens long). We check if the big model needs to be halted by verifying if any of the completions contain `</big` (to indicate `</bigmodel>`, since **we do not make these control tokens special tokens**). While this naive implementation *is still faster than running just the big model*, it wastes several prefill steps that can be hidden, and decode checks that are not necessary. We write a simple simulation code that

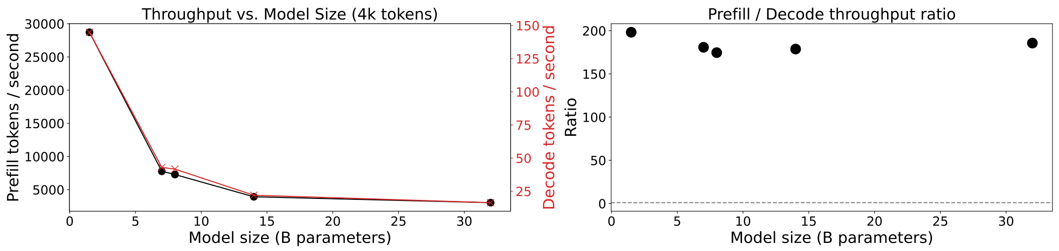


Figure 8: **(Left)** Prefill and Decode throughput decreases drastically as model size increases. Decoding *most tokens* from a small model will drastically improve end-to-end latency. **(Right)** Prefill can be upto 200× faster given sufficient input sequence length, further, large model prefill is *still faster* than small model decode ( 3000 tokens / sec vs. 150 tokens / sec). This indicates that the large model will be able to keep up with the small model generation.

models the pipelined decode behavior in Figure 2. Our inference simulation numbers in Figure 4 are generated by profiling models of sizes `1.5B, 7B, 8B, 14B, 32B, 70B` on A6000 GPUs to feed the appropriate prefill and decode throughput numbers to our simulator. We present our prefill and decode throughput in Figure 8. Given a sufficiently long input sequence, prefill is significantly faster than decode. While a small model (1.5B) is over $8\times$ faster at decode than a big model (32B) it is still slower than the big model (32B) prefill, indicating that our proposed pipelined inference flow is feasible. From Figure 1,

## 5 Discussion

**Alignment with Latency:** In this paper, we propose to use control tokens (`<bigmodel>`) and latency-aware feedback (in the GRPO reward formulation) to demonstrate that it is possible to use **RL for alignment with hardware**, not just human preferences. This gives rise to several interesting questions on how to leverage control tokens to teach a model to optimize its own inference (**RL4L**). This could be in forms beyond just offloading, such as quantization, pruning, compression.

**Limitations:** We primarily focus on keeping an efficient training flow, this means our GRPO formulation does not actually offload the generation to the big-model when the small-model emits a `<bigmodel>` token. We instead rely on prompting and SFT to roughly *indicate* that these tags must be placed in difficult regions. However, with more compute resources, significantly better offloading behavior may be induced by actually modeling true offloading latency in the reward formulation, as well as true offloading for accuracy evaluation. Further, our current performance measurements are *simulation based*. While the simulation is very straightforward, implementing controlling prerfill and streaming prefills to study the implications of *chunk size* of prefill is important, and may impact latency. Further, our method still requires huge Key-Value caches, as both the small and big model have to retain the KV-Cache. Further, we require more devices (GPUs) to run both the big and small model. These costs are similar to that of speculative decoding, but still merit further discussion in light of recent innovations in KV-Cache sharing, dynamic model pruning etc.

## 6 Conclusion

We introduce SpeculativeReasoners, a class of models that can *learn to offload* their own decode, with the goal of optimizing performance and accuracy. This is a novel optimization to reasoning models, where we aim to use RL for latency (**RL4L**), aligning language models for *performance*. Our early results indicate that by just offloading 1-5% of the decode process to a larger model, AIME24 accuracy can increase by 20-30%. Surprisingly, even random-offloading can boost accuracy. We also find that even using a relatively small model (8B parameters) as the *big* model can lead to a huge increase in model quality.

## References

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. URL `https://arxiv.org/abs/2005.14165`.

[2] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022. URL `https://arxiv.org/abs/2204.02311`.

[3] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023. URL `https://arxiv.org/abs/2302.13971`.

[4] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022. URL `https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf`.

[5] Mingyu Jin, Qinkai Yu, Dong Shu, Haiyan Zhao, Wenyue Hua, Yanda Meng, Yongfeng Zhang, and Mengnan Du. The impact of reasoning step length on large language models. *arXiv preprint arXiv:2401.04925*, 2024.

[6] Charles Condevaux and Sébastien Harispe. Lsg attention: Extrapolation of pretrained transformers to long sequences. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 443–454. Springer, 2023.

[7] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. URL `https://arxiv.org/abs/2307.08691`.

[8] Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, Beidi Chen, Guangyu Sun, and Kurt Keutzer. Llm inference unveiled: Survey and roofline model insights, 2024. URL `https://arxiv.org/abs/2402.16363`.

[9] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023. URL `https://arxiv.org/abs/2203.11171`.

[10] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL `https://arxiv.org/abs/2201.11903`.

[11] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

[12] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[13] Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL `https://arxiv.org/abs/2302.01318`.

[14] Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of llm inference using lookahead decoding, 2024. URL `https://arxiv.org/abs/2402.02057`.

[15] Nikhil Bhendawade, Irina Belousova, Qichen Fu, Henry Mason, Mohammad Rastegari, and Mahyar Najibi. Speculative streaming: Fast LLM inference without auxiliary models, 2025. URL `https://openreview.net/forum?id=jt8wI3ZzXG`.

[16] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads, 2024. URL `https://arxiv.org/abs/2401.10774`.

[17] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning, 2022. URL `https://arxiv.org/abs/2203.14465`.

[18] Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning, 2024. URL `https://arxiv.org/abs/2409.12917`.

[19] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL `https://arxiv.org/abs/2305.20050`.

[20] Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (rest) for language modeling, 2023. URL `https://arxiv.org/abs/2308.08998`.

[21] Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search, 2024. URL `https://arxiv.org/abs/2406.03816`.

[22] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao

Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL `https://arxiv.org/abs/2501.12948`.

[23] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL `https://arxiv.org/abs/2402.03300`.

[24] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL `https://arxiv.org/abs/2303.17651`.

[25] Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning, 2025. URL `https://arxiv.org/abs/2503.04697`.

[26] Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. URL `https://arxiv.org/abs/2501.19393`.

[27] Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL `https://github.com/huggingface/open-r1`.

[28] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023. URL `https://zenodo.org/records/10256836`.