LANEROPE: POSITIONAL ENCODING FOR COLLABORATIVE PARALLEL REASONING AND GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Parallel LLM test-time scaling techniques (e.g., best-of-N) require drawing $N > 1$ sequences conditioned on the same input prompt. These methods boost accuracy while exploiting the computational efficiency of batching N generations. However, each sequence in the batch is traditionally generated independently and hence does not reuse intermediate generations, computations, or observations from other sequences. In this paper, we propose LaneRoPE to enable coordination and collaboration among $N > 1$ sequences at generation time. LaneRoPE involves two key ideas: (a) an inter-sequence attention mask to make sampling of sequences dependent on one another; and (b) a RoPE extension that injects positional information that captures relative positions between tokens, both within and outside a particular sequence. We evaluate our approach on mathematical reasoning tasks and find promising results: LaneRoPE enables collaboration among sequences, yielding additional accuracy gains under limited generated sequence length. Importantly, since LaneRoPE enables coordination with minimal changes to the underlying LLM architecture and introduces negligible overhead at inference time, it is appealing to rapidly incorporate parallel reasoning into existing LLM inference pipelines.

1 INTRODUCTION

Parallel test-time scaling methods improve LLM performance with additional inference-time compute instead of finetuning (Cobbe et al., 2021; Brown et al., 2024; Wu et al., 2025; Snell et al., 2024; Wang et al., 2022). The key idea of these methods is generating *multiple* responses (each potentially with its own reasoning trace) for a given input, and optionally, ranking and aggregating responses with a reward model. The multiple responses are typically generated in parallel during the same forward pass (i.e., batched inference), and benefit from a hardware efficiency standpoint, since this amortizes memory transfers, minimizes kernel launch overheads, and better utilizes compute cores at each generation step. Despite its efficiency, parallel inference is largely *uncoordinated*: for a given input, the output sequences are generated independently. This prohibits exploiting inherent decomposable problem structures (Yang et al., 2025), limits the sharing of intermediate outputs between sequences, and the generation of diverse responses.

Building on powerful parallel inference, how can we make multiple sequences coordinate with each other during generation? Two families of approaches have gained traction, each with their inherent trade-offs. The first family of approaches leverage 'explicit branching' (Gandhi et al., 2024; Pan et al., 2025; Yang et al., 2025; Zheng et al., 2025) mechanisms: at inference time, a single sequence adaptively fans-out to multiple search paths (e.g., one path per decomposed subproblem) and funnels them into a single response. Although they encourage the LLM to explicitly decompose tasks, the dynamic nature of how variable number of parallel processes are spawned and merged introduces challenges in predicting runtime compute resources and more importantly, they depend on specialized run-time engines. The second family of 'collaborative' approaches (Hsu et al., 2025; Rodionov et al., 2025; Dong et al., 2025), under which our approach falls, introduce a more flexible means of coordination primarily by enabling shared visibility (Fig. 1a) into intermediate generations of all parallel sequences. These techniques are practically attractive, since they rely on a fixed number of generation sequences (i.e., constant batch size; see Fig. 1b) and can hence largely leverage batching support by both LLM inference engines and the underlying hardware. In particular, existing techniques (Hsu et al., 2025; Rodionov et al., 2025) exploit an interesting insight that multiple sequences can be 'virtually ordered' into a single sequence by reparameterizing

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

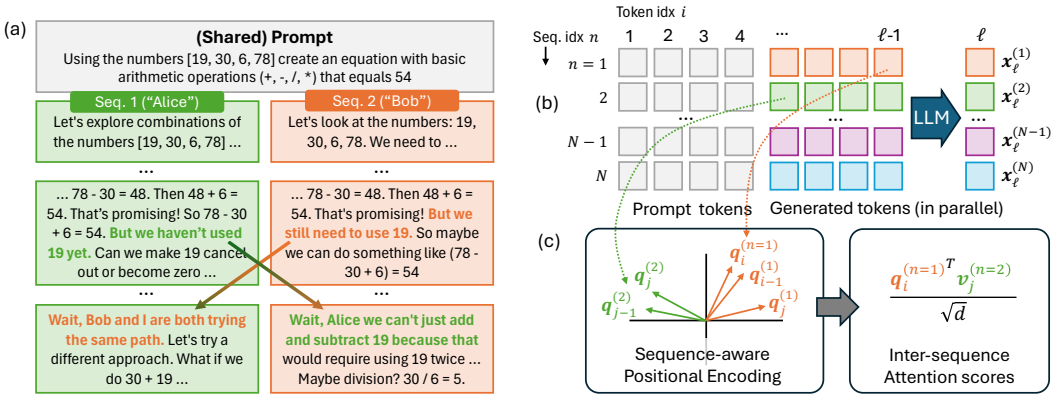


Figure 1: **LaneRoPE: High-level introduction and intuition.** (a) We investigate the problem of *collaborative reasoning*, where given a single input prompt, parallel sequences can reason by conditionally attending to other sequences mid-generation; (b) Our work assumes the tokens across sequences are generated in parallel using the same model, and thereby benefit from *batched efficiency*; (c) A key contribution of our work is introducing *cross-lane attention*, such that attentions can be calculated across sequences (i.e., *lanes*). To enable this, we introduce a novel position encoding scheme to account for relative distances between tokens of different sequences.

the positions or positional encodings. However, we argue that this is inherently a multi-sequence generation problem and inference must accurately reflect positions both along the token dimension (as typically done) *and* sequence (or *lane*) dimension. In the rest of this manuscript, we use *lane* and *sequence* interchangeably to refer to batch elements generated simultaneously and inter-dependently.

In this work, we propose LaneRoPE to better enable parallel reasoning and generation. Similar to parallel test-time scaling techniques (Wang et al., 2022; Wu et al., 2025) and collaborative reasoning (Hsu et al., 2025; Rodionov et al., 2025; Dong et al., 2025), we assume a constant number of sequences (Fig. 1b) generated at inference. Our approach involves two key insights. First, we make intermediate reasoning traces and outputs visible across sequences by conditioning the next token on previous tokens from *all sequences*. We achieve this by using causal cross-sequence attention masks and generalizing attention score calculations to extend to inter-sequence tokens. Second, we extend rotary positional encoding 'RoPE' (Su et al., 2021) to additionally capture relative distances among sequences (Fig. 1b), allowing us to learn the attention structure across parallel generations in a dynamic and task-dependent manner. Our formulation is expressive enough to capture a range of parallel inference techniques, ranging from independent sampling (e.g., self-consistency) to the fully dense attention masks used in GroupThink (Hsu et al., 2025).

Our evaluation of LaneRoPE on standard mathematical reasoning datasets (e.g., AIME, MATH500) and two open-weight LLMs shows promising results. Overall, we find that the LLMs demonstrate a reliable accuracy improvement across multiple benchmarks. The flexibility in our formulation enables LaneRoPE to be integrated into existing models with minimal modification and, importantly, in contrast to prior work (Hsu et al., 2025; Rodionov et al., 2025), admits finetuning. Furthermore, finetuning requires very few additional parameters ($< 0.5\%$ over base models), which can be fit with simple SFT and synthetic data. In summary, we find LaneRoPE takes a promising step towards enabling parallel *collaborative* reasoning by fundamentally exploiting relative positions of tokens, both within and outside a particular sequence.

Contributions We summarize our contributions as:

- A novel method to enable *fine-grained token-level collaboration* during batched parallel generation of language models. Notably, our method encompasses some existing techniques but its superior flexibility and generality allow for further fine-tuning.
- A recipe to synthesize *collaborative reasoning traces*, used for supervised fine-tuning (SFT).
- A fine-tuning pipeline to unlock and strengthen the parallel reasoning capabilities of pre-trained models.

2 Related Work

LLMs, Reasoning and Test-time Search. Large language models (LLMs) have attracted significant attention, largely due to their remarkable performance across a range of tasks. Many of these tasks (e.g., math, coding) test the LLM's ability for generalization, abstraction, and multi-step problem solving. Especially for such tasks, reasoning (Wei et al., 2022; Kojima et al., 2022) has proven to be a powerful paradigm and has significantly improved success rates at solving challenging problems. These techniques can be further extended to explore various reasoning paths before concluding with a final solution. The reasoning search paths can take various structures, but can broadly be classified into sequential (Muennighoff et al., 2025), graph-like (Yao et al., 2023; Besta et al., 2024), or parallel (Wang et al., 2022; Brown et al., 2024). In this work, we primarily study reasoning as a parallel search problem.

Parallel Reasoning: Independent and Inter-dependent. A popular strategy to scale compute at test-time to improve accuracies is with parallel generations (Wang et al., 2022; Cobbe et al., 2021). For a given problem, multiple sequences (potentially chain-of-thought reasoning followed by a solution) from the same LLM are sampled in parallel and ranked (e.g., using a reward model) to determine the final solution. Prior works (Wang et al., 2022; Cobbe et al., 2021; Brown et al., 2024; Snell et al., 2024; Wu et al., 2025) primarily demonstrate effectiveness where the sequences are sampled *independently*. This however leads to inefficient use of compute, since there is little collaboration between sequences and risks redundant compute between sequences. A new recent line of work explores whether the sequences can be sampled *inter-dependently* (Dong et al., 2025), such that LLMs can decide to adaptively parallelize and coordinate *during* generation. This line of inter-dependent generation broadly falls under two categories: (a) 'adaptive branching' (Jin et al., 2025; Pan et al., 2025; Qi et al., 2025; Yang et al., 2025; Chen et al., 2025; Zheng et al., 2025; Wen et al., 2025; Lian et al., 2025) where at generation time, special tokens (e.g., `fork` and `merge`) signal the inference engine to spawn or merge multiple parallel sequence generations; and (b) 'fine-grained collaboration', as in Hsu et al. (2025); Rodionov et al. (2025) and recently Dong et al. (2025), where sequence generation is always parallel (constant batch size) and each sequence can attend to part or all of the tokens generated by other sequences using dynamic attention masks. Our approach falls into the second category, and unlike previous works we introduce a flexible framework to positionally encode cross-sequence tokens which additionally account for relative inter-sequence distances.

Positional Encoding in LLMs Large language models (LLMs) incorporate *positional encodings* to inject sequence order information into transformer architectures, since the self-attention mechanism alone is permutation-invariant (Vaswani et al., 2017). The original Transformer used fixed sinusoidal position encodings (Vaswani et al., 2017), while subsequent models like BERT adopted learned positional embeddings (Devlin et al., 2019), both providing each token with an absolute position bias. To improve generalization beyond fixed-length contexts, *relative positional encodings* were introduced to encode pairwise token distances directly in self-attention (Shaw et al., 2018), allowing the model to focus on relative order. More recent approaches modify how positional information enters the attention mechanism to enable better extrapolation to longer sequences. For example, *rotary positional encodings* (RoPE) rotate query/key vectors by angle proportional to token index, effectively combining absolute and relative positional information (Su et al., 2021). Similarly, *ALiBi* (Attention with Linear Biases) eliminates explicit embeddings and instead adds a distance-dependent bias to attention scores, allowing models trained on shorter sequences to generalize to much longer inputs (Press et al., 2022). In our work, we extend RoPE (Su et al., 2021) to additionally model relative distances between sequences.

3 LaneRoPE: Cross-lane positional encoding

In this section, we present our approach LaneRoPE to enable collaborative parallel inference. We begin by covering preliminaries, and then we detail our approach.

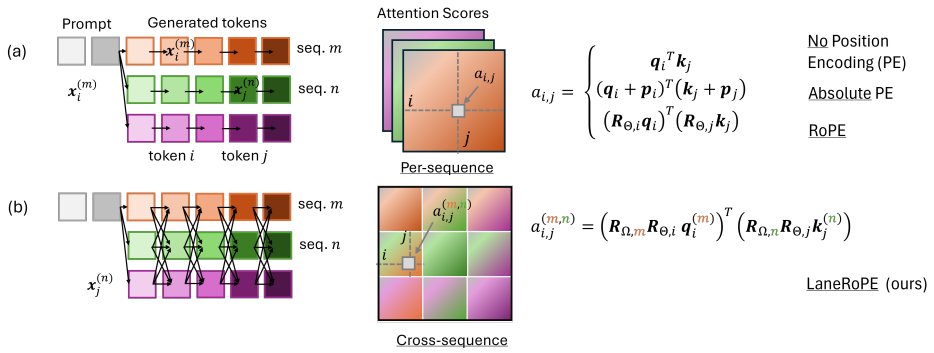


Figure 2: Comparison of RoPE (Su et al., 2021) and LaneRoPE for parallel inference. (a) Parallel inference (e.g., best-of-N) relies on generating sequences independently. As a result, positional encodings and attention scores are defined only per-sequence. (b) In LaneRoPE, tokens are generated by causally attending to tokens from all sequences. We achieve this by introducing cross-sequence attention scores with a novel position encoding scheme to account for inter-sequence relative distances.

3.1 Preliminaries

A standard self-attention layer (Vaswani et al., 2017) takes token embeddings $\{\boldsymbol{x}_i\}_{i=1}^{L}$, computes queries, keys and values (Eq. 1) and produces corresponding attention logits and outputs (Eq. 2):

$$\boldsymbol{q}_i = f_q(\boldsymbol{x}_i, i), \quad \boldsymbol{k}_j = f_k(\boldsymbol{x}_j, j), \quad \boldsymbol{v}_j = f_v(\boldsymbol{x}_j, j) \tag{1}$$

$$\alpha'_{i,j} = \mathrm{softmax}_j\left(\boldsymbol{q}_i^\top \boldsymbol{k}_j / \sqrt{d}\right), \quad \boldsymbol{o}_i = \sum_{j=1}^{L} \alpha'_{i,j} \boldsymbol{v}_j \tag{2}$$

Positional Encodings and RoPE Positional encodings injected into $f_{\{q,k\}}$ make outputs position-dependent. RoPE (Su et al., 2021) enforces that attention logits depend only on the relative offset $i - j$ by rotating embeddings with a block-diagonal matrix $\boldsymbol{R}_{\Theta,i}$ (see Apx. A):

$$f_{\{q,k\}}(\boldsymbol{x}_i, i) = \boldsymbol{R}_{\Theta,i}\left(\boldsymbol{W}_{\{q,k\}}\boldsymbol{x}_i + \boldsymbol{b}_{\{q,k\}}\right) \tag{3}$$

$$\boldsymbol{R}_{\Theta,i} = \mathrm{diag}\big(R(i\theta_1), \ldots, R(i\theta_{d/2})\big) \qquad \text{with frequencies params.: } \Theta = (\theta_l)_{l=1}^{d/2} \tag{4}$$

$$R(i\theta_\ell) = \begin{bmatrix} \cos(i\theta_\ell) & \sin(i\theta_\ell) \\ -\sin(i\theta_\ell) & \cos(i\theta_\ell) \end{bmatrix} \in \mathbb{R}^{2\times 2} \qquad \ell = 1, \ldots, d/2 \tag{5}$$

Parallel Test-time Scaling Parallel test-time scaling (Brown et al., 2024; Snell et al., 2024; Wu et al., 2025) improves LLM accuracy at inference time by sampling $N > 1$ sequences and selecting the best via majority vote (Wang et al., 2022) or a verifier (Cobbe et al., 2021; Lightman et al., 2024). Most prior work assumes *independent* sampling across sequences.

3.2 LaneRoPE: Sequence-aware Positional Encoding

Multi-sequence generation We generate $N > 1$ sequences $\{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$ auto-regressively in parallel for the same prompt:

$$p(\boldsymbol{x}_{i+1}^{(n)} \mid \texttt{prompt}, \{\boldsymbol{x}_{1:i}^{(m)}\}_{m=1}^{N}) \tag{6}$$

Here, as opposed to *independent* sampling (conditional generation of $\boldsymbol{x}_{i+1}^{(n)}$ on $\boldsymbol{x}_{1:i}^{(n)}$), we now have *inter-dependent* generation (conditioned on $\{\boldsymbol{x}_{1:i}^{(m)}\}_{m=1}^{N}$) via *causal inter-sequence attention*, allowing tokens in sequence m to attend to tokens in sequence n:

$$\alpha'^{(m,n)}_{i,j} = \mathrm{softmax}_{j,n}\left(\boldsymbol{q}_i^{(m)\top} \boldsymbol{k}_j^{(n)} / \sqrt{d}\right) \tag{7}$$

However, since tokens at the same time step across sequences share identical positional indices, standard RoPE cannot distinguish their sequence identities.

Cross-sequence Positional Encoding To distinguish between sequence identities, we introduce a second block-diagonal rotation $\boldsymbol{R}_{\Omega,m}$ that encodes the sequence index m:

$$\boldsymbol{q}_i^{(m)} = f_q(\boldsymbol{x}_i^{(m)}, i, m), \quad \boldsymbol{k}_j^{(n)} = f_k(\boldsymbol{x}_j^{(n)}, j, n) \tag{8}$$

$$f_{\{q,k\}}(\boldsymbol{x}_i, i, m) = \boldsymbol{R}_{\Omega,m}\boldsymbol{R}_{\Theta,i}(\boldsymbol{W}_{\{q,k\}}\boldsymbol{x}_i^{(m)} + \boldsymbol{b}_{\{q,k\}}) \tag{9}$$

with lane frequencies $\Omega = \{\omega_t\}_{t=1}^{d/2}$. The associativity $R(\omega_t m)R(\theta_t i) = R(\omega_t m + \theta_t i)$ lets both rotations be merged and applied together, enabling drop-in integration into any RoPE architecture. Analogously, while RoPE uses a 1D Fourier basis over the token sequence, LaneRoPE uses a 2D Fourier basis over the joint (token, sequence) grid. An important effect of $\boldsymbol{R}(\Omega_t, m)$ being orthogonal is that the (pre-softmax) attention weight between two tokens i, j within the same sequence m is unchanged since both key and query vectors are rotated by $\boldsymbol{R}(\Omega_t, m)$ and, therefore, the inner product ${\boldsymbol{q}_i^{(m)}}^\top \boldsymbol{k}_j^{(m)}$ remains the same. In particular, for $N = 1$ lane, LaneRoPE simplifies to standard RoPE.

3.3 LaneRoPE: Properties and Initialization Strategies

Thanks to its flexibility, LaneRoPE can express some existing parallelization strategies, but can also go beyond if trained. This suggests using (a combination of) these strategies to *initialize LaneRoPE from pre-trained models*, in order to leverage their native reasoning and cooperation capabilities.

Special Case: GroupThink GroupThink (Hsu et al., 2025) previously proposed causal inter-sequence attention by 'virtually' ordering generation of multiple sequences, albeit within a single sequence. We find this to be a special case of our proposed approach and elaborate further. GroupThink distinguishes sequences by assigning token $\boldsymbol{x}_i^{(m)}$ the virtual position $Km + i$ (sequence gap K), so that within-sequence tokens appear close in RoPE-space while cross-sequence tokens appear further away. This is exactly recovered in LaneRoPE via the rotation-matrix decomposition

$$\begin{aligned}\boldsymbol{R}_{\Theta,Km+i} &= \boldsymbol{R}_{\Theta,Km}\boldsymbol{R}_{\Theta,i} \\ &= \boldsymbol{R}_{K\Theta,m}\boldsymbol{R}_{\Theta,i} = \boldsymbol{R}_{\Omega,m}\boldsymbol{R}_{\Theta,i}\end{aligned} \tag{10}$$

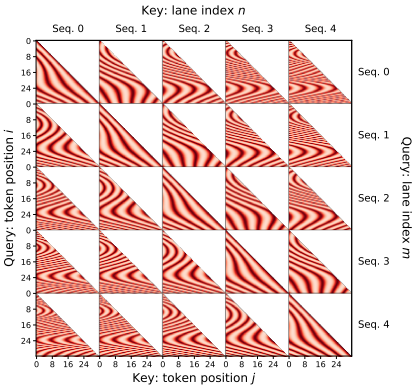
with $\Omega := \{K\theta_t\}$, i.e. GroupThink is LaneRoPE with lane frequencies $\Omega = K\Theta$. A *limitation* is that tokens from higher-index sequences ($n > m$) appear at *negative* virtual relative positions when attended by sequence m, which is out-of-distribution for causally pre-trained models.[1]

Special Case: Parallel Independent Sampling Independent sampling across N sequences is equivalent to adding a sequence-identity bias $\beta(m-n) \propto \delta_{[m=n]}$ to the attention logits, so that cross-lane scores are suppressed:

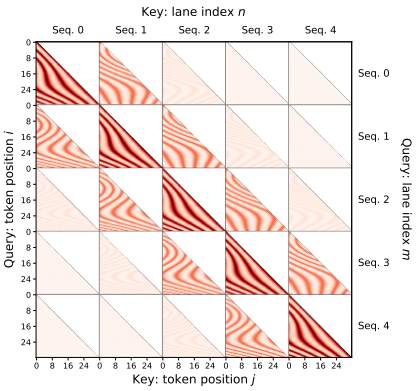
$$a_{i,j}^{(m,n)} = {\boldsymbol{q}_i^{(m)}}^\top \boldsymbol{k}_j^{(n)} + \beta(m-n) \tag{11}$$

We approximate β via a F-dimensional discrete Fourier basis and incorporate it into LaneRoPE by augmenting projection layers with F more bias dimensions (see Apx. C):

$$\boldsymbol{W}'_{\{q,k\}} := \begin{pmatrix} \boldsymbol{W}_{\{q,k\}} \\ \boldsymbol{O} \end{pmatrix}, \quad \boldsymbol{b}'_{\{q,k\}} := \begin{pmatrix} \boldsymbol{b}_{\{q,k\}} \\ \hat{\boldsymbol{\beta}} \end{pmatrix} \tag{12}$$



(a) $\beta(0) = 0$, i.e. $|\hat{\boldsymbol{\beta}}|_2 = 0$



(b) $\beta(0) > 0$, i.e. $|\hat{\boldsymbol{\beta}}|_2 > 0$

Figure 3: Effect of the attention bias β on attention scores in LaneRoPE, controlled by $|\hat{\boldsymbol{\beta}}|_2$. A GroupThink initialization is used, where tokens from different lanes occupy distinct virtual position indices. The causal mask within each lane is preserved.

[1] Hogwild! (Rodionov et al., 2025) avoids this by dynamically reordering sequences, but requires custom attention kernels.

with $\boldsymbol{O} \in \mathbb{R}^{F \times d_{\text{emb}}}$ zero-initialized. This leaves all pre-trained weights intact and adds negligible overhead ($\approx$25M parameters for F=4 in `Qwen2.5-7B`, i.e. a 0.34% increment). The bias strength is controlled by $|\hat{\boldsymbol{\beta}}|_2$; Fig. 3 shows its effect on the resulting attention pattern.

Initialization Strategy: NTK-aware correction for GroupThink To mitigate the *negative virtual indexes limitation* of GroupThink, we draw inspiration from the "NTK-aware" interpolation methods like YaRN (Peng et al., 2025) which modify RoPE's frequencies to extend the context length of pre-trained models. The main idea is that, despite the translation equivariance of RoPE, frequencies with a relatively long period compared to the pre-trained context length enable the model to encode absolute positions, rather than relative ones. We suspect these low frequencies suffer the most from the negative relative indexes of GroupThink, which fall outside the portion of the period typically observed during pre-training. Hence, we introduce a scaling function $\gamma(\theta_t) \in [0, 1]$ which smoothly decays to 0 frequencies below a certain threshold, i.e. $\Omega := \{\gamma(\theta_t) K \theta_t\}_t$; this limits the lane index effect on them, retaining their capacity to represent absolute positions; see Apx. C for more details.

In practice, we combine these strategies: lane frequencies $\boldsymbol{\omega}' = K\Theta$ (GroupThink or NTK) for cross-lane structure, and the Fourier bias for initialization to independent sampling. Since $\boldsymbol{W}'_{\{q,k\}}$ and $\boldsymbol{b}'_{\{q,k\}}$ are fully learnable, training is free to move beyond these special cases.

4 Training LaneRoPE

To elicit cross-lane interactions in a base model, we consider two fine-tuning strategies: *i)* supervised finetuning (SFT) on a dataset that mimics collaborations, and *ii)* optimizing with a training objective adapted from Kahneman-Tversky Optimization (KTO) (Ethayarajh et al., 2024) on a dataset of (un)successful reasoning traces independently generated by the base model.

SFT with collaborative data We construct this dataset using questions from the *DeepScaleR-Preview-Dataset* (Luo et al., 2025) with `Qwen3-30B-A3B-Instruct-2507` (Qwen-Team, 2025). To simulate collaboration across N lanes using a sequential LLM, we prompt the LLM to play the role of N different assistants in a step-by-step manner. An assistant's message at each step ends after 128 new tokens. After each step, an assistant is prompted to generate a new message by completing its own answer, pre-filled with the text it generated so far, which follows a user message providing the query and the text previously generated by other assistants. For the final SFT dataset, we separate the text generated by each assistant into individual traces such that each trace only contains the messages of one assistant. As a result, a model fine-tuned on this dataset requires a form of cross-sequence interaction to be able to fit the data. In total, we generate 3 conversations with $N = 2, 3$ or 4 assistants on a subset of questions of the dataset, resulting in about 50k raw conversations. See Apx. G for more details. To focus on quality over quantity, we further filter the original generations based on the following criteria. First, we only keep samples where all the assistants give the correct answer. Second, we employ a length limit in range $10 - 30$ messages per lane and remove all samples outside that range. Third, we use a simple heuristic to filter out conversations with little or no interactions between the assistants. After this curation process, the final SFT dataset consists of 4797 conversations. See Apx. G.1 for an example of the SFT dataset.

KTO with independent reasoning traces We sample 8 completions with the base model (either `DeepSeek-R1-Distill-Qwen-1.5B` or `DeepSeek-R1-Distill-Qwen-7B`) and a maximum completion length of 4096, for each of the 40315 questions of the *DeepScaleR-Preview-Dataset* (Luo et al., 2025). Among these, we keep only those queries where *at most half* of the answers were correct, to ensure a sufficiently challenging dataset. Importantly, we preserve both good and wrong completions to be used for the KTO loss (Ethayarajh et al., 2024), which respectively maximizes and minimizes their likelihoods, up to some per-sample saturation. Hence, we create a dataset by grouping the completions of each query into subsets of $N = 2, 3, 4$ lanes, ensuring each subset contains at least one correct answer, for about 15k total groups; during training, LaneRoPE is able to cross-attend among the N lanes within each subset. While SFT with independently sampled data would force our model to mimic the i.i.d. distribution, batches with mixed good/wrong completions can *encourage a non-trivial joint distribution among parallel lanes via the KTO loss*: e.g. it's easier to minimize the likelihood of a wrong completion when its final answer doesn't match that of the other completions in the batch. See Apx. E.

Other training details To train our models, we first initialize them from open-source pre-trained LLMs using a combination of the initialization strategies described earlier. In particular, we use a "sequence gap" $K = 8192$ to bias the model towards a GroupThink-like behavior but also set a strong positional bias $|\hat{\boldsymbol{\beta}}|_2^2 = 1000$ to encourage the model to attend to tokens in other sequences only when necessary. Then, we train our models for one epoch on these curated datasets. In our experiments with small $1.5B$, we train all their weights while we use LoRA (including trainable biases) for the $7B$ ones to reduce memory cost at training time and fit at least 4 lanes within a single GPU. When indicated with a $*$, we also train the frequencies Ω of the LaneRoPE itself. See Apx. E.

5 Experiments

To assess the effectiveness of our proposed method, we conduct evaluations across several reasoning benchmarks and models that measure diverse problem-solving competencies. We conclude commenting on the efficiency of LaneRoPE, but refer to Apx. F for detailed profiling.

Specifically, we benchmark performance on the MATH500 (Lightman et al., 2024), the AMC 23 (Mathematical Association of America, 2023), and AIME 24 and 25 (Mathematical Association of America, 2024; 2025) datasets, all of which consist of challenging mathematical problems. We test our method on the fine-tuned reasoning models `DeepSeek-R1-Distill-Qwen` with 1.5B and 7B parameters (DeepSeek-AI, 2025). To compare to existing work, we utilize the official implementation of Hogwild! (Rodionov et al., 2025) and evaluate it on the same datasets and models as LaneRoPE. Due to lack of public code and to avoid potential differences in hyperparameters and evaluation implementation (see also Apx. D.1), we further implement a Bridge (Dong et al., 2025) baseline ourselves, which we train and evaluate like our models. In our experiments, each multi-lane model is tested over different $N \geq 1$ parallel sequences. We sample M sequences for each query in each benchmark: these are sampled independently in standard baselines but, when using N-lanes models, we independently sample M/N groups of N inter-dependent completions; see Apx. E. Finally, all generations use a temperature of 0.6 and `top-p` value of 0.95 (as recommended in DeepSeek-AI (2025)) and are limited by 4096 maximum tokens. All training runs were conducted on Nvidia A100 GPUs, with a combination of single- and multi-GPU training runs on single nodes.

5.1 Results

In Tab. 1, we compare the `maj@4` score across different mathematical reasoning datasets. We report `maj@4` scores as they provide a fair metric to compare models operating with different numbers N of lanes, since all models have the same fixed parallel budget $B = 4$; see also discussion and other scores in Apx. D.1. We first note that all trained 7B LaneRoPE models show consistent improvements over the sequential reasoning baseline [DS-Qwen-7B]. It is interesting to see that this is occasionally true also for the *untrained* variants, initialized with GroupThink or its NTK corrected version; however, we also find these models to suffer significant loss in average accuracy (see Tab. 3): indeed, these models generate many corrupted completions, but these don't affect the majority count (as no boxed answer is generated) and the resulting `maj@4` score. This artifact disappears in trained models, which not only retain but also increase the average accuracy in Tab. 3.

Next, we find KTO training to be more effective than SFT. This likely reflects the larger dataset used, which is the result of the simpler data-generation and filtering pipeline behind the KTO training strategy. A more detailed *ablation study*, comparing the different initialization and training choices for LaneRoPE, can be found in Tab. 4 in appendix; in particular, the best configuration is confirmed to be the NTK-aware initialization with trainable LaneRoPE frequencies and KTO training.

In Tab. 1, we also compare our implementation of Bridge (Dong et al., 2025). Bridge employs an "axial" cross-lanes attention: the N tokens simultaneously generated attend to each other but they don't share the KV cache. With respect to Dong et al. (2025), who employ a combination of SFT and RL to reinforce parallel reasoning capabilities, we train Bridge with the same SFT and KTO settings of LaneRoPE for fair comparison. LaneRoPE typically outperforms Bridge on 7B models, reflecting its higher capacity to share information between parallel sequences. Interestingly, we find different scaling patterns in the SFT regime, where data contains explicit collaborative traces: ***LaneRoPE consistently gains accuracy with multiple lanes, while Bridge loses performance***. This pattern also replicates with the `pass@4` score in Tab. 2, as well as with the 1.5B models. These results *isolate the*

Model	Train.	N	MATH500	AIME24	AIME25	AMC23	Avg.
DS-Qwen-1.5B	-	1	79.2	19.4	19.1	56.5	43.5
	KTO	1	84.6	22.1	25.0	73.1	51.2
+HogWild	-	2	58.6	7.5	6.6	43.3	27.2
+LaneRoPE(GT)	-	2	43.7	0.0	0.0	20.2	16.0
+Bridge	KTO	1	86.9	32.1	26.6	74.2	54.9
		2	86.4	33.4	25.5	73.9	54.8
		4	87.0	28.7	27.9	73.0	54.1
	SFT	1	82.0	22.0	24.7	63.4	48.0
		2	80.9	19.7	19.5	66.2	46.6
		4	81.4	23.3	20.4	62.6	46.9
+LaneRoPE(NTK*)	KTO	1	84.5	28.3	24.6	72.3	52.4
		2	83.9	26.3	25.2	71.3	51.7
		4	84.6	26.3	25.6	68.7	51.3
	SFT	1	81.5	20.7	20.4	62.2	46.2
		2	79.8	21.9	26.5	62.1	47.6
		4	81.6	18.2	20.4	68.6	47.2
DS-Qwen-7B	-	1	86.3	25.7	26.1	70.2	52.1
	KTO	1	90.8	45.0	31.4	84.0	62.8
	SFT	1	88.0	37.1	28.1	76.8	57.5
+HogWild	-	2	70.4	20.8	21.1	58.0	21.8
+LaneRoPE(GT)	-	2	85.0	27.1	27.9	70.2	52.5
+LaneRoPE(NTK)	-	2	84.3	28.2	24.1	65.2	50.5
+Bridge	KTO	1	85.5	39.4	29.4	82.8	59.3
		2	88.8	40.5	29.2	81.4	60.0
		4	89.4	40.0	33.2	85.0	61.9
	SFT	1	88.6	34.5	30.2	75.6	57.2
		2	88.3	31.8	30.6	77.0	56.9
		4	87.1	29.9	31.0	71.6	54.9
+LaneRoPE(NTK*)	KTO	1	90.5	44.4	34.4	82.7	**63.0**
		2	91.5	46.5	33.3	84.2	**63.9**
		4	90.7	46.3	33.6	85.9	**64.1**
	SFT	1	87.3	32.6	28.5	76.7	56.3
		2	89.0	36.0	29.1	75.9	57.5
		4	87.8	34.1	28.5	81.1	57.9

Table 1: `maj@4` score averaged over 12 samples per query for MATH500 and 16 samples per query across other mathematical reasoning benchmarks. All other numbers are our independent runs. GT = GroupThink initialization, NTK = our NTK-aware correction to GroupThink, * = learnable LaneRoPE frequencies, - = no additional training (pre-trained). Bold text highlight the highest performance for each number of lanes N.

benefits of attending to the lanes' full history, which Bridge can't do, necessary to fully exploit the collaborative SFT data.

The smaller 1.5B model shows weaker results, not benefiting from (untrained) GroupThink. Training generally recovers the performance, even improving over the [DS-Qwen-1.5B] baseline. While the SFT-ed model benefits from $N > 1$ lanes, KTO training breaks this pattern. A similar trend is observed in the Bridge 1.5B baseline, indicating a limitation of the small base model, rather than the architecture. We note that the observed greater benefits for the larger 7B model are in line with the findings in Rodionov et al. (2025), i.e. that larger models have better collaboration capabilities.

We find that Hogwild! tends to worsen the base model performance. This does not warrant the conclusion that it is an ineffective method, but this discrepancy suggests that certain factors may affect performance, most notably the use of a different underlying base model than the one originally evaluated. For example, we observe Hogwild! produces typically rather verbose answers which might indicate under-performance at the limited completion length of 4096. Such change may alter the dynamics of the Hogwild! prompt and require significant additional tuning to achieve better results.

Finally, we question whether LaneRoPE is more effective than self-consistency-based strategies like *majority voting* when considering larger parallel generation budgets k. To answer this we study combinations of LaneRoPE with *majority voting* in Fig. 4. Again, we observe that the 7B model shows consistent gains from using inter-dependent samples via LaneRoPE, rather than independent samples.

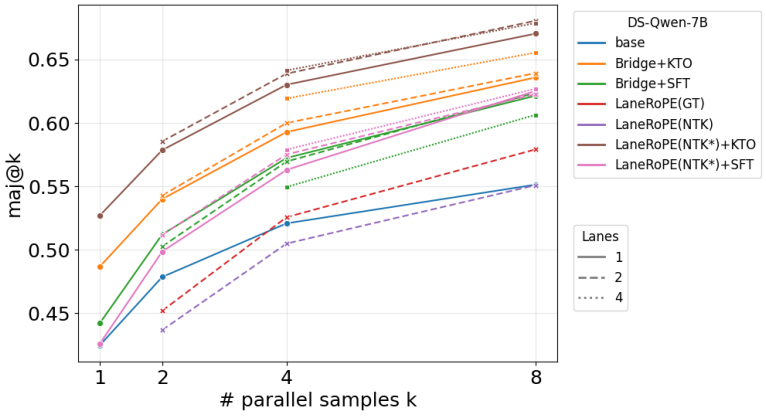


Figure 4: Avg. `maj@`k performance across the datasets as a function of the parallelization budget k.

Inference time efficiency An important benefit of LaneRoPE is its negligible overhead. Indeed, inference can be performed by interleaving the input tokens into a N times longer sequence (carefully tracking indexes) and, then, using standard attention back-ends; see Apx. B for more details on the practical implementation. Apx. F includes detailed profiling to support our efficiency claim. In particular, while cross-lane attention among N lanes increases by a factor of N the attention layers' FLOPs, LLM decoding is typically *memory-bound*, rather than compute-bound, implying additional attention FLOPs do not necessary translate into inference overhead. This is the case in our setting, too. Indeed, our profiling measurements in Apx. F (Tab. 6) show negligible impact on inference time with respect to the standard model. We believe these results justify the adoption of LaneRoPE as an efficient and lightweight solution to enable inter-sequence collaboration.

6 Conclusion

In this paper, we addressed the problem of collaborative parallel inference, where multiple sequences (e.g., within the same batch of a forward pass) can access each other's partial generation to condition future reasoning. In particular, we proposed a novel and flexible method - which encompasses some previous approaches - to model token-level inter-sequence interactions efficiently, along with a data-generation recipe to create parallel and collaborative reasoning traces for fine-tuning. Our empirical results show improved reasoning capabilities in parallel setups and open up opportunities for better parallel scaling strategies beyond independent sampling.

Limitations and Future Work We do not yet employ a dedicated training scheme for merging outputs from multiple lanes, and developing such mechanisms remains an open area for refinement. Our current experiments focus on $N \leq 4$ collaborating lanes, though the method itself is designed to extend naturally to a larger number of lanes. While LaneRoPE increases context length by a factor of N, we found this to have negligible impact on performance in the small batch regime - relevant for inference on the edge - in Sec. F. Finally, while a small SFT or KTO stage with synthetic data proved sufficient to unlock parallel reasoning capabilities, it also proved to be the main bottleneck to further scale our method. Additional refinement stage with Reinforcement Learning with Verifiable Feedback (RLVF), as in Dong et al. (2025), could unlock novel collaborative behaviors and strategies in an unsupervised way and further strengthen the overall reasoning performance.

AI USE STATEMENT

In this work, we used generative AI tools to generated the synthetic dataset, as described in Apx. G. Beyond this, we used AI tools to proofread and improve the manuscript quality and help with formatting problems. Finally, AI tools were used to assist writing the code for precise profiling and to aggregate and plot the results of the experiments.

REPRODUCIBILITY STATEMENT

We report details about the hardware used, the training details and the hyperparameters chosen in Sec. 5 and Apx. E. Apx. G also reports details on how we generated the synthetic dataset.

REFERENCES

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, volume 38, pp. 17682–17690, 2024.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*, 2024.

Keyu Chen, Zhifeng Shen, Daohai Yu, Haoqian Wu, Wei Wen, Jianfeng He, Ruizhi Qiao, and Xing Sun. Aspd: Unlocking adaptive serial-parallel decoding by exploring intrinsic parallelism in llms. *arXiv preprint arXiv:2508.08895*, 2025. doi: 10.48550/arXiv.2508.08895. URL https://arxiv.org/abs/2508.08895.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4171–4186, 2019.

Harry Dong et al. Generalized parallel scaling with interdependent generations. *arXiv preprint arXiv:2510.01143*, 2025. URL https://arxiv.org/abs/2510.01143.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*, 2024.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D Goodman. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*, 2024.

Nathan Habib, Clémentine Fourrier, Hynek Kydlíček, Thomas Wolf, and Lewis Tunstall. Lighteval: A lightweight framework for llm evaluation, 2023. URL https://github.com/huggingface/lighteval.

Chan-Jan Hsu, Davide Buffelli, Jamie McGowan, Feng-Ting Liao, Yi-Chang Chen, Sattar Vakili, and Da-shan Shiu. Group think: Multiple concurrent reasoning agents collaborating at token level granularity. In *arXiv preprint arXiv:2505.11107*, 2025. doi: 10.48550/arXiv.2505.11107. URL https://arxiv.org/abs/2505.11107.

Tian Jin, Ellie Y. Cheng, Zack Ankner, Nikunj Saunshi, Jonathan Ragan-Kelley, Suvinay Subramanian, Blake M. Elias, Amir Yazdanbakhsh, and Michael Carbin. Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding. In *Proceedings of the*

42nd International Conference on Machine Learning (ICML), 2025. URL https://icml.cc/virtual/2025/poster/44837.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022.

Long Lian et al. Threadweaver: Adaptive threading for efficient parallel reasoning in language models. *arXiv preprint arXiv:2512.07843*, 2025. URL https://arxiv.org/abs/2512.07843.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. https://huggingface.co/agentica-org/DeepScaleR-1.5B-Preview, 2025.

Mathematical Association of America. American mathematics competitions (amc) 10/12 2023, 2023. URL https://maa.org/.

Mathematical Association of America. American invitational mathematics examination (aime) 2024, 2024. URL https://maa.org/.

Mathematical Association of America. American invitational mathematics examination (aime) 2025, 2025. URL https://maa.org/.

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. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20286–20332, Suzhou, China, November 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

Jiayi Pan, Xiuyu Li, Long Lian, Charlie Snell, Yifei Zhou, Adam Yala, Trevor Darrell, Kurt Keutzer, and Alane Suhr. Learning adaptive parallel reasoning with language models. In *Conference on Language Modeling (COLM)*, 2025. URL https://arxiv.org/abs/2504.15466.

Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. In *The Twelfth International Conference on Learning Representations*, 2025.

Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. In *International Conference on Learning Representations (ICLR)*, 2022.

Jianing Qi, Xi Ye, Hao Tang, Zhigang Zhu, and Eunsol Choi. Learning to reason across parallel samples for llm reasoning. *arXiv preprint arXiv:2506.09014*, 2025. doi: 10.48550/arXiv.2506.09014. URL https://arxiv.org/abs/2506.09014.

Qwen-Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Erik Schultheis, Vage Egiazarian, Anton Sinitsin, Denis Kuznedelev, and Dan Alistarh. Hogwild! inference: Parallel llm generation via concurrent attention. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2504.06261.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Vol. 2: Short Papers)*, pp. 464–468, 2018.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NIPS 2017)*, pp. 6000–6010, 2017.

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. *arXiv preprint arXiv:2203.11171*, 2022.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. Parathinker: Native parallel thinking as a new paradigm to scale llm test-time compute. *arXiv preprint arXiv:2509.04475*, 2025. doi: 10.48550/arXiv.2509.04475. URL https://arxiv.org/abs/2509.04475.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. Inference scaling laws: An empirical analysis of compute-optimal inference for llm problem-solving. In *The Thirteenth International Conference on Learning Representations*, 2025.

Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. Multiverse: Your language models secretly decide how to parallelize and merge generation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. URL https://arxiv.org/abs/2506.09991.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. Flashinfer: Efficient and customizable attention engine for llm inference serving. *arXiv preprint arXiv:2501.01005*, 2025. URL https://arxiv.org/abs/2501.01005.

Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, Runpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, and Dong Yu. Parallel-r1: Towards parallel thinking via reinforcement learning. *arXiv preprint arXiv:2509.07980*, 2025. doi: 10.48550/arXiv.2509.07980. URL https://arxiv.org/abs/2509.07980.

A Background: Self-attention and Positional Encodings

Self-attention The self-attention layer of a standard transformer (Vaswani et al., 2017) takes as inputs a set of tokens $\mathbb{T} = \{t_i\}_{i=1}^{L}$. Each token is represented by its corresponding embedding $\mathbb{E} = \{\boldsymbol{x}_i\}_{i=1}^{L}$, where $\boldsymbol{x}_i$ is a d^{emb}-dim embedding of the i-th token without positional information. The self-attention layer converts embeddings into query, key, and value representations via a linear transformation over positionally-encoded embeddings (Eq. 1), with attention logits $a_{i,j} = \boldsymbol{q}_i^\top \boldsymbol{k}_j$ and output $\boldsymbol{o}_i = \sum_j \alpha'_{i,j} \boldsymbol{v}_j$ (Eq. 2).

Positional Encodings and RoPE Positional encodings are injected at the query/key level so that attention outputs are position-dependent. A traditional choice (Vaswani et al., 2017) uses *absolute* positional encodings $f_{\{q,k\}}(\boldsymbol{x}_i, i) = \boldsymbol{W}_{\{q,k\}}(\boldsymbol{x}_i + \boldsymbol{p}_i)$, with $\boldsymbol{p}_{i,2t} = \cos\left(\frac{i}{10000^{2t/d}}\right)$ and $\boldsymbol{p}_{i,2t+1} = \sin\left(\frac{i}{10000^{2t/d}}\right)$. However, most token interactions are *translation-invariant*, motivating attention scores that depend on relative offsets. RoPE (Su et al., 2021) enforces this by splitting dimensions into $d/2$ independent 2D planes, rotating each with frequency θ_ℓ:

$$f_q(\boldsymbol{x}_i, i)^T f_k(\boldsymbol{x}_j, j) = g(\boldsymbol{x}_i, \boldsymbol{x}_j, i - j) \tag{13}$$

where the rotation matrices (Eq. 4) use per-plane rotation blocks:

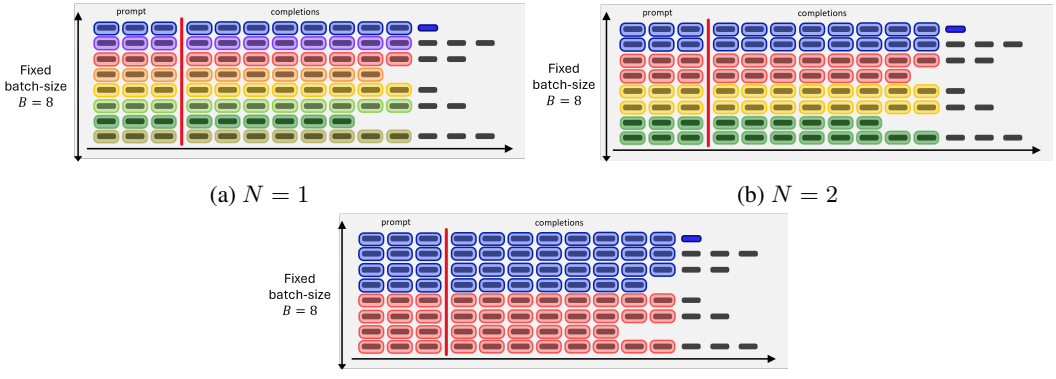
$$R(i\theta_\ell) = \begin{bmatrix} \cos(i\theta_\ell) & \sin(i\theta_\ell) \\ -\sin(i\theta_\ell) & \cos(i\theta_\ell) \end{bmatrix} \quad \ell = 1, \ldots, d/2 \tag{14}$$

such that the key-query inner product satisfies

$$\boldsymbol{q}_i^\top \boldsymbol{k}_j = (\boldsymbol{W}_q \boldsymbol{x}_j + \boldsymbol{b}_q) \boldsymbol{R}_{\Theta, i-j} (\boldsymbol{W}_k \boldsymbol{x}_i + \boldsymbol{b}_k) \,. \tag{15}$$

Multi-sequence generation notation We generate $N > 1$ sequences $\{\boldsymbol{x}^{(n)}\}_{n=1}^{N}$ in parallel, where the n-th sequence is an ordered list of tokens $\boldsymbol{x}^{(n)} = [\boldsymbol{x}_1^{(n)}, \boldsymbol{x}_2^{(n)}, \ldots]$. The standard assumption generates each sequence independently from the prompt and its own past tokens. LaneRoPE instead conditions each token on all sequences' past tokens (Eq. 6), implemented via a modified causal attention mask that exposes the full KV cache of all lanes to each query token.

B Precise attention mask and Lanes packing



(a) $N = 1$ (b) $N = 2$

(c) $N = 4$

Figure 5: Visual representation of the effective attention masks employed when adopting cross-lanes attention across different number N of lanes. Note that the total batch size B and, therefore, the total number of tokens and KV cache size remain unchanged. In the image, the next blue token attends to all tokens in the KV cache highlighted in blue; changing N only changes the number of tokens attended to. Fig. 6 shows how this cross-lanes attention is efficiently implemented in practice.

In all our experiments, we maintain the memory usage (the effective batch size B, the KV cache size and the number of queries) fixed when varying the number of lanes N. In other words, different values

of N typically affect only the mask employed in the attention layers. Fig. 5 includes a visualization to clarify this, representing the effect of choosing different numbers N of lanes on the attention mask.

Importantly, Fig. 5 only provides an abstract representation of the attention masks. To efficiently implement LaneRoPE, we follow the strategy represented in Fig. 6. In practice, we reshape the KV cache (and the query tokens) of N parallel lanes to leverage standard causal attention implementations. This has the practical benefit of reducing the computational overhead of the N-times more attention FLOPs to a negligible effect on the inference time; see analysis and profiling in Apx. F.

Another side effect of this implementation choice is that the N tokens generated by N lanes at the generation step t also attend to each other with a causal attention mask, i.e. at time step t the token $x_t^{(i)}$ attends only to the tokens $x_t^{(j)}$ of its previous lanes $j < i$ (but still attends to all tokens $x_{t'}^{(j)}$ for $t' < t$ and any $j < N$). While this implementation detail is not necessary for LaneRoPE, it is a natural choice which makes the adoption of standard causal attention backends seamless.

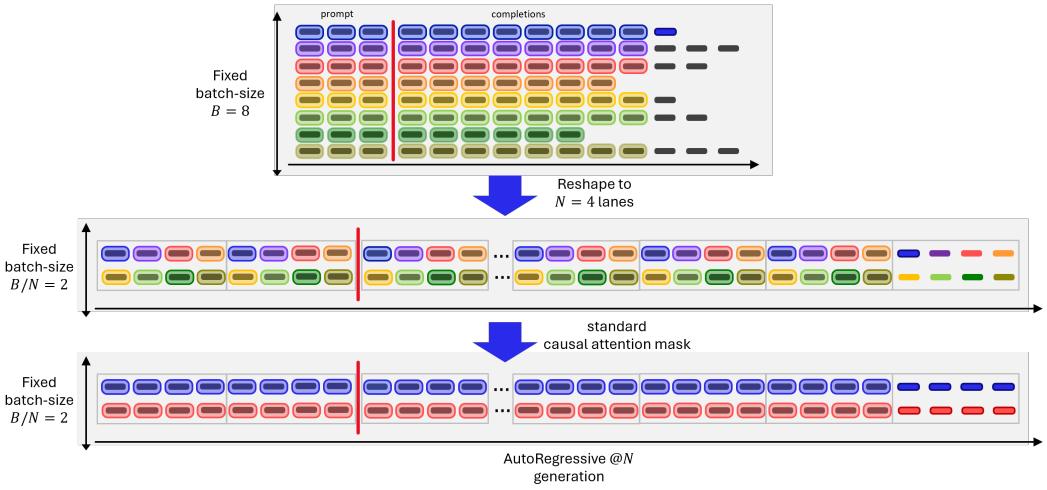


Figure 6: Visual representation of how LaneRoPE's cross-lanes attention is efficiently implemented using standard causal attention. In particular, note that at each generation step, the N next tokens belonging to a group of N lanes also attend to each other with a causal mask; while this is not necessary for LaneRoPE, it is a convenient implementation choice which enables a seamless adoption of efficient standard causal attention backends.

C Initialization Strategies: Additional Details

Independent Sampling via Fourier Attention Bias We approximate the identity-selector $\beta(x) \propto \delta_{[x=0]}$ using a F-dimensional discrete Fourier basis over $F/2$ frequencies $\boldsymbol{\omega} = \{\omega_t\}_{t=1}^{F/2}$:

$$\beta(x) := \sum_{t=1}^{F/2} \hat{\beta}_t^c \cos(2\pi\omega_t x) + \hat{\beta}_t^s \sin(2\pi\omega_t x) \tag{16}$$

Since β is symmetric, $\hat{\beta}_t^s = 0$ and all $\hat{\beta}_t^c \geq 0$. Defining $\hat{\boldsymbol{\beta}} \in \mathbb{R}^F$ with coefficients $\{\sqrt{\hat{\beta}_t^{\{c,s\}}}\}_t$ lets us write $\beta(x) = \hat{\boldsymbol{\beta}}^\top R_{\boldsymbol{\omega},x}\, \hat{\boldsymbol{\beta}}$, which is an inner product of rotated vectors - exactly the form produced by LaneRoPE on the augmented dimensions.

We incorporate this into LaneRoPE by augmenting query/key projections as in Eq. 12, with correspondingly enlarged rotary matrices:

$$R_{\Theta',i} := \begin{pmatrix} R_{\Theta,i} & \\ & \boldsymbol{I} \end{pmatrix}, \quad R_{\Omega',m} := \begin{pmatrix} R_{\omega',m} & \\ & R_{\omega,m} \end{pmatrix} \tag{17}$$

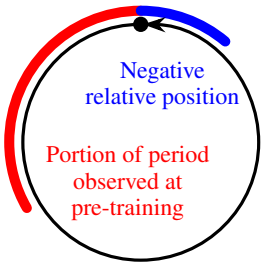
where $R_{\Theta,i}$ is the pre-trained RoPE matrix and $R_{\omega',m}$ can be any LaneRoPE matrix (we use $\boldsymbol{\omega}' = K\Theta$ following GroupThink). The resulting projection has the same form as Eq. 9:

$$f_{\{q,k\}}(\boldsymbol{x}_i, i, m) = \boldsymbol{R}_{\Omega',m}\boldsymbol{R}_{\Theta',i}(\boldsymbol{W}'_{\{q,k\}}\boldsymbol{x}_i^{(m)} + \boldsymbol{b}'_{\{q,k\}}) \tag{18}$$

Note that LaneRoPE is also sufficiently expressive to represent β without the additional dimensions, but the augmented form avoids modifying any pre-trained weights. For N parallel sequences, $N+1$ additional dimensions always suffice for an exact Fourier representation over the sequence indices.

When combining this strategy with GroupThink, we simply set the lane frequencies $\boldsymbol{\omega}' = K\Theta$ (or according to the NTK-aware correction $\boldsymbol{\omega}' = \gamma \cdot K\Theta$, as below).

NTK-aware correction for GroupThink As observed by the authors of YaRN Peng et al. (2025), some RoPE dimensions have a frequency θ_t with a wavelength $\lambda_t = \frac{2\pi}{\theta_t}$ larger than (or close to) the pre-training context size L, and, therefore there are rotation angles in these dimensions which were never or rarely observed during training by the model, as represented on the right. Conversely, higher frequencies have shorter wavelength and the full range of rotations was observed often during training. We argue these higher frequencies are more robust to GroupThink since they properly model relative positions and since negative relative positions still represent rotations which were sufficiently observed during training.



Conversely, the low frequencies model the distance from the beginning of the sequence and are more sensitive to GroupThink indexing. YaRN defines a ramp function $\gamma \in [0, 1]$ to weight and interpolate smoothly between two choices: interpolation by scaling the low frequencies and preserving high frequencies. Similarly, we use a ramp function γ to interpolate between zeroing out the low frequencies (s.t. positional embedding depends only on the sequence index, but not on the lane index) and applying GroupThink on the highest ones. At dimension t, define the ratio $r_t = \frac{L}{\lambda_t}$ where λ_t is the wavelength and L the pre-traning max context size. Then, the ramp function is

$$\gamma(r_t) = \begin{cases} 0 \text{ if } r_t < \alpha \\ 1 \text{ if } r_t > \beta \\ \frac{r-\alpha}{\beta-\alpha} \text{ otherwise} \end{cases}$$

and the final LaneRoPE frequency at dimension t is

$$\omega_t := \gamma(r_t) \cdot K\theta_t$$

where K is the sequence gap of GroupThink. We use $\alpha = 4$ and $\beta = 32$.

D Additional Experiments

D.1 Pass@4 metrics and Accuracy

For completeness, in this section we also report the `Pass@4` and Accuracy (`Pass@1`) metrics of the models we compared on different mathematical reasoning benchmarks in the main paper. In particular, Tab. 2 reports the `Pass@4` while Tab. 3 reports the Accuracy (`Pass@1`).

However, we highlight an important difference of the Accuracy (`Pass@1`) metrics with the `maj@4` metric we reported in Table 1 in the main paper: accuracy measures the average performance of each individual completion, hence models with $N > 1$ lanes have an implicit advantage, being able to attend to $N > 1$ reasoning traces when producing each individual answer. In contrast, Table 1 provides a more fair comparison where each model has the same constant budget of 4 parallel completions, aggregated via majority voting.

Comparison with Bridge Dong et al. (2025) We could not reproduce the accuracy (`Pass@1`) scores of the baseline `DeepSeek-R1-Distill-Qwen-1.5B` and `DeepSeek-R1-Distill-Qwen-7B` models reported by the authors of Bridge in Dong et al. (2025). In particular, comparing the results in Table 3 with the ones reported in Table 1 of

Model	Train.	N	MATH500	AIME24	AIME25	AMC23	Avg.
DS-Qwen-1.5B	-	1	80.0	19.6	19.5	56.6	43.9
	KTO	1	86.5	24.2	26.8	75.2	53.2
+LaneRoPE(GT)	-	2	45.6	0.0	0.0	21.2	16.7
+Bridge	KTO	1	88.8	36.1	28.7	78.7	58.1
		2	88.8	35.7	28.1	77.9	57.6
		4	89.0	31.7	28.3	76.2	56.3
	SFT	1	86.9	26.6	29.6	71.1	53.6
		2	84.9	22.1	23.0	73.4	50.9
		4	84.5	25.0	27.5	68.1	51.3
+LaneRoPE(NTK*)	KTO	1	86.4	32.7	26.2	76.4	55.4
		2	86.5	30.6	25.5	74.1	54.2
		4	86.9	28.3	27.5	73.8	54.1
	SFT	1	84.6	23.1	24.8	66.5	49.7
		2	83.1	25.1	28.7	65.6	50.6
		4	84.9	20.8	22.5	73.1	50.3
DS-Qwen-7B	-	1	86.6	26.0	26.1	70.2	52.2
	KTO	1	92.1	45.3	32.6	84.4	63.6
	SFT	1	89.9	38.7	32.4	79.8	60.2
+LaneRoPE(GT)	-	2	87.8	29.4	30.0	72.9	55.0
+LaneRoPE(NTK)	-	2	87.0	32.7	25.1	68.5	53.3
+Bridge	KTO	1	90.4	41.9	32.3	85.2	62.4
		2	90.4	41.5	32.1	82.9	61.8
		4	90.4	40.0	34.2	85.0	62.4
	SFT	1	90.7	36.2	32.3	79.1	59.6
		2	90.4	33.0	33.8	79.0	59.0
		4	89.5	35.0	32.5	75.6	58.2
+LaneRoPE(NTK*)	KTO	1	91.5	45.8	36.1	83.6	64.2
		2	92.3	47.9	35.5	85.0	65.2
		4	91.7	46.7	35.0	86.2	64.9
	SFT	1	88.9	35.1	33.1	80.3	59.4
		2	90.8	37.7	32.3	78.1	59.7
		4	89.9	36.7	33.3	83.1	60.8

Table 2: `Pass@4` score averaged over 12 samples per query for MATH500 and 16 samples per query across other mathematical reasoning benchmarks. All other numbers are our independent runs.

Dong et al. (2025), our evaluations of *our base models* show lower accuracies (e.g. 76.5 vs 82.15 for `DeepSeek-R1-Distill-Qwen-7B` on MATH500), despite our attempts to use the same evaluation parameters (temperature 0.6, `top-p=` 0.95 and 4096 max generated tokens). This suggests other differences in evaluation which make a direct comparison with their reported results misleading. For this reason, instead of reporting the numbers from Dong et al. (2025), we chose to implement Bridge ourselves (following the details in the Dong et al. (2025)) to train and evaluate it with the same setup of our models.

D.2 Ablation Study

Table 4 reports an ablation study to determine the effect the different initialization choices of LaneRoPE have and compare the two training strategies we proposed. Overall, KTO training shows

consistently better results than SFT. The best results are achieved with the NTK initialization and learnable frequencies, supporting the benefits of LaneRoPE's flexible design.

Model	Train.	N	MATH500	AIME24	AIME25	AMC23	Avg.
DS-Qwen-1.5B	-	1	66.3	10.2	10.6	43.1	32.6
	KTO	1	76.5	14.6	17.9	58.9	42.0
+HogWild	-	2	42.4	2.5	3.3	21.2	17.4
+LaneRoPE(GT)	-	2	17.3	0.0	0.0	8.1	6.4
+Bridge	KTO	1	79.2	19.2	17.7	62.0	44.5
		2	79.1	20.6	18.1	60.5	44.6
		4	78.9	19.4	18.5	59.5	44.1
	SFT	1	72.8	13.1	15.0	52.0	38.2
		2	70.6	11.2	12.9	50.9	36.4
		4	70.5	13.8	13.5	50.0	36.9
+LaneRoPE(NTK*)	KTO	1	76.7	16.9	18.5	59.4	42.9
		2	76.2	14.6	19.0	59.4	42.3
		4	75.9	15.6	18.1	57.7	41.8
	SFT	1	67.9	10.8	12.3	47.8	34.7
		2	67.7	11.5	15.8	48.6	35.9
		4	68.8	10.4	13.3	51.6	36.0
DS-Qwen-7B	-	1	76.5	17.5	19.8	55.9	42.4
	KTO	1	84.3	29.6	25.8	71.1	52.7
	SFT	1	75.4	21.7	19.6	58.0	43.7
+HogWild	-	2	28.7	5.0	1.7	14.4	12.4
+LaneRoPE(GT)	-	2	45.6	11.7	11.7	33.1	25.5
+LaneRoPE(NTK)	-	2	44.6	12.2	10.0	32.1	24.7
+Bridge	KTO	1	78.1	26.0	24.4	66.1	48.7
		2	80.9	25.8	23.1	66.7	49.2
		4	82.3	27.1	25.6	65.5	50.1
	SFT	1	76.3	20.8	19.8	59.8	44.2
		2	76.2	17.7	20.6	59.4	43.5
		4	74.0	17.7	19.4	55.2	41.6
+LaneRoPE(NTK*)	KTO	1	84.9	29.0	26.2	70.6	52.7
		2	84.9	29.8	26.0	70.5	52.8
		4	82.2	31.0	25.0	72.8	52.8
	SFT	1	75.1	17.7	19.2	58.3	42.6
		2	77.3	21.5	20.4	59.8	44.7
		4	74.5	20.6	18.5	60.3	43.5

Table 3: Accuracy (`Pass@1`) score averaged over 12 samples per query for MATH500 and 16 samples per query across other mathematical reasoning benchmarks. All other numbers are our independent runs. GT = GroupThink initialization, NTK = NTK-aware correction to GroupThink, * = learnable LaneRoPE frequencies. Note that Accuracy (`Pass@1`) measures the average performance of each individual completion, hence models with $N > 1$ lanes have an implicit advantage since each answer leverage $N > 1$ reasoning traces. Here, we report Accuracy (`Pass@1`) for completeness but refer to Table 1 for a more fair comparison using `maj@4`, where each model has the same constant budget of 4 parallel completions.

Model	Train.	N	AIME24	AIME25	AMC23	Avg
DS-Qwen-7B		1	25.7	26.1	70.2	40.7
+LaneRoPE(GT)	KTO	1	42.4	29.9	78.3	50.2
		2	39.3	30.2	78.7	49.4
		4	46.2	33.8	82.5	<u>54.2</u>
	SFT	1	33.0	30.6	72.4	45.3
		2	29.6	28.0	73.5	43.7
		4	32.0	24.6	78.3	45.0
+LaneRoPE(GT*)	KTO	1	41.2	32.7	76.1	50.0
		2	43.2	32.4	71.6	49.1
		4	40.0	33.3	77.5	50.3
	SFT	1	34.5	30.1	76.5	47.0
		2	33.2	27.2	76.6	45.7
		4	34.8	25.4	71.7	44.0
+LaneRoPE(NTK)	KTO	1	42.1	32.5	84.4	<u>53.0</u>
		2	45.9	32.0	83.7	<u>53.9</u>
		4	44.5	33.6	83.1	53.8
	SFT	1	32.3	31.0	73.8	45.7
		2	31.5	27.2	74.4	44.4
		4	39.0	28.8	74.9	47.5
+LaneRoPE(NTK*)	KTO	1	44.4	34.4	82.7	**53.8**
		2	46.5	33.3	84.2	**54.7**
		4	46.3	33.6	85.9	**55.3**
	SFT	1	32.6	28.5	76.7	45.9
		2	36.0	29.1	75.9	47.0
		4	34.1	28.5	81.1	47.9

Table 4: Ablation Study: maj@4 score of different variations of a `DeepSeek-R1-Distill-Qwen-7B` -LaneRoPE model. The models are initialized with either GroupThink or the NTK-aware correction and then trained with the SFT or the KTO strategy. * indicates whether the LaneRoPE frequencies are trainable or not. We evaluate with maj@4 for fair comparison between number N of lanes. Overall, KTO training shows consistently better results. The best results are achieved with the NTK initialization and learnable frequencies, supporting the benefits of LaneRoPE's flexible design.

E Additional Experiments Details

Model hyper-parameters While LaneRoPE introduces a few hyperparameters, like the additional frequencies dimensions F for the attention bias, the scale of this bias $|\hat{\boldsymbol{\beta}}|_2$ or the sequence gap K. In practice, these hyperparameters have clear meanings which guide their choice, so we don't find the need for explicitly optimizing them. In all our experiments, we use 2 frequencies (i.e. $F = 4$ additional dimensions), as that is the minimum needed to represent arbitrary functions on $N = 4$ lanes via discrete Fourier transform. For the *sequence gap*, we choose $K = 8192$ as it exceeds the longest single-lane sequence (prompt + at most 4096 generated tokens), so lanes cannot share virtual token positions at initialization (recall the GroupThink analogy). Finally, the bias magnitude is set to $|\hat{\boldsymbol{\beta}}|_2^2 = 1000$, large enough to initialize the model to independent sampling, i.e. such that likelihoods stay close to the base model's at initialization, which keeps the KTO objective centered at the start of training. These configurations result in the following parameters counts:

- DS-Qwen-7B (original): 7.616B parameters. DS-Qwen-7B + LaneRoPE ($F = 4$): 7.641B parameters. Increment: ≈ 25.7M params (0.34%)
- DS-Qwen-1.5B (original): 1.777B parameters. DS-Qwen-1.5B + LaneRoPE ($F = 4$): 1.782B parameters. Increment: ≈ 4.8M params (0.27%).

In both cases, the additional learnable LaneRoPE frequencies only contribute 66 parameters.

SFT Training We train all models for one epochs on our collaborative datasets with a total batch size of 60, (each containing groups of $N = 2, 3, 4$ collaborative reasoning traces). We initialize the other LaneRoPE frequencies either like GroupThink as $\Omega = K\Theta$ or like the NTK-corrected variant, by using a *sequence gap* $K = 8192$. We train all our 7B models using a LoRA adapter (with rank 32 and including the bias of the key and query linear projection layers) to reduce memory cost at training time and fit at least 4 lanes within a single GPU, with a learning rate of 1e-4 and a batch size of 60. Due to the large initialization norm $|\hat{\boldsymbol{\beta}}|_2^2 = 1000$ in the biases of the attention linear layers, we typically use a larger learning rate of 1e-2 only for these parameters. Similarly, we adopt a stronger learning rate of 1e-2 for the LaneRoPE frequency parameters Ω, when tuning these parameters. We also adopt a cosine learning rate scheduler with a warmup ratio of 10% and weight decay 5e-2 (excluding the LaneRoPE frequency parameters Ω).

KTO training The KTO loss has form

$$v(x, y) = \begin{cases} \lambda_D \sigma(\beta(\log \pi - \log \pi_{\text{ref}} - z_0)) \text{ for positive example } y \\ \lambda_U \sigma(-\beta(\log \pi - \log \pi_{\text{ref}} - z_0)) \text{ for negative example } y \end{cases} \tag{19}$$

where π is the policy model, i.e. the model we want to train and π_{ref} is a reference model. λ_U and λ_D are hyper-parameters controlling the weights of undesirable and desirable examples, for non-balanced datasets (we use $\lambda_D = 1$, $\lambda_U = 0.7$). β is a hyperparameter (we use $\beta = 0.1$) that controls the strength of the regularization with the reference model. With respect to Ethayarajh et al. (2024), we do not use the KL term z_0 since it is more expensive to compute and the models are initialized to independent sampling and, therefore, already similar to the model that generated the data. In our experiments, whenever we adapt a base model with LaneRoPE, we always use the base model as a *reference* model π_{ref} in the KTO loss. Moreover, we replace the sigmoid σ with an asymmetric function σ' defined as

$$\sigma'(x) = \begin{cases} \sigma(x) & \text{if } x \geq 0 \\ -x + \frac{1}{2} & \text{otherwise} \end{cases} \tag{20}$$

which ensures saturation only on one side. This is because, despite the independent sampling initialization as in Sec. C, LaneRoPE can give slightly lower likelihood estimations than the original model π_{ref} (especially the small 1.5B model). A symmetric saturating loss there would be undesirable since it would prevent any gradient at the beginning of the training until $\log \pi$ matches precisely $\log \pi_{\text{ref}}$.

For training, we use a learning rate of 1e-6 and a batch size of 72 for the 1.5B models, where we tune all weights, and a learning rate of $2e-5$ and batch size 40 for the 7B models, where we adopt LoRA adapters (rank 32 and including query and key linear layers' biases). Like in the SFT experiments, we use a larger learning rate of 1e-2 for the query and key linear layers' biases and, when tuned, the LaneRoPE frequency parameters Ω. We also adopt a cosine learning rate scheduler with a warmup ratio of 10% and weight decay 5e-2 (excluding the LaneRoPE frequency parameters Ω). Each batch contains completions randomly grouped in $N = 2, 3$ or 4 parallel lanes, with the total number of completions roughly constant (different values of N approximately share the same number of total completions, which means groups of large N sequences appear relatively less frequently in the data).

We adopted similar configurations when training Bridge models; however, we observed degraded performance when using LoRA adapters in the 7B (with identity-initialized Bridge layers), hence we opted for training only its Bridge layers (as done in the warm-up phase in Dong et al. (2025)) with a reduced learning rate of $1e-6$. Note these models contain roughly ≈ 200M trainable parameters, versus ≈ 90M of the LoRA LaneRoPE equivalent model. We tune all the weights for 1.5B models.

Evaluation In all our experiments, we instruct the models to provide an answer within `\boxed{}`. Hence, for each sequence, we extract the answer from the last occurrence of `\boxed{}` in the completion and, then, use the Lighteval framework (Habib et al., 2023) to score it.

When computing the majority voting scores in Fig. 4, from each sample of k sequences, we filter out sequences that do not contain any `\boxed{}` before computing the majority. When using LaneRoPE (or other interdependent methods like Hogwild! and Bridge) models with $N > 1$ lanes and a budget of k parallel sequences, we sample k/N independent groups of N inter-dependent sequences, i.e. maintaining the same batch size overall, as visualized in Apx. B.

When estimating `maj@4` or `pass@4` scores (i.e. $k = 4$), we compute them by resampling from the pool of $M = 16$ completions generated for each query. To be precise, for an $N \leq 4$ model, we have M/N independent groups of completions and we randomly sample $P \gg M$ subsets of size $k = 4$ (i.e. $4/N$ groups of interdepentent completions). Then, we compute the `maj@4` (or `pass@4`) score for each of these P groups and report the average. When feasible, we use all possible $P = \binom{M/N}{4/N}$ such subsets, otherwise we rely on random sampling.

All generations use a temperature of 0.6 and `top-p` value of 0.95 (as recommended in DeepSeek-AI (2025)) and are limited by 4096 maximum tokens.

F Inference time measurements

An important benefit of LaneRoPE is that it comes with little-to-no overhead over standard transformer architectures. Indeed, inference can be done by interleaving the input tokens into a N times longer sequence and then using standard attention backends; see also Apx. B. In this section, we include some detailed profiling measurements to support this claim.

Lanes N	**@256**	**@2304**	**@4352**	**@6400**	**@8448**
1 / base	0.8 / 122.6	7.4 / 129.2	14.0 / 135.8	20.6 / 142.4	27.1 / 148.9
2	1.6 / 123.4	14.8 / 136.6	28.0 / 149.8	41.1 / 162.9	54.3 / 176.1
4	3.3 / 125.1	29.6 / 151.4	55.9 / 177.7	82.2 / 204.0	108.5 / 230.3
4 - base	2.5	22.2	41.9	61.6	81.4

Table 5: Estimated FLOPs (GFLOP) of a forward pass for a `DeepSeek-R1-Distill-Qwen-7B` at *total* batch size $B = 8$ and at different context sizes. $N > 1$ uses multi-lane attention within groups of N out of B sequences. Each cell reports two numbers, *attention-only / whole-model FLOPs* (in GFLOP). The dense term is constant (≈ 121.8 GFLOP): during decoding the projections and MLP act only on the B new tokens, not on the cache, and a step emits B new tokens for any value of N, so only attention depends on context length and lane count N. In practice, decoding is memory rather than compute-bound in our setting, so these increased FLOP counts don't affect the measured inference time, as Tab. 6 shows.

First of all, of course the additional context introduced by the cross-lanes attention increases the FLOPs of the model. Tab. 5 reports the estimated FLOPs[2] per inference step of a `DeepSeek-R1-Distill-Qwen-7B` base model ($F = 0$) when we enable attention to different number of lanes N, while keeping the batch size B (and, therefore, the total memory usage and KV cache size) fixed (see Fig. 5). The effect of the increased attention context is initially negligible with respect to the full model's inference, but plays a gradually major role as the context size increases.

However, LLM inference typically happens in *memory-bound* regimes and, therefore, the additional attention FLOPs do not necessary translate into measured inference overhead. This is the case when the device is bottle-necked by loading the weights and the KV cache in memory to compute the attention layer's output, rather than by the computations in the layer. To verify this, we implemented a `flashInfer`-based (Ye et al., 2025) inference backend that performs LaneRoPE attention directly over a paged KV cache. Then, we measured device-only decode latency by seeding paged FlashInfer KV caches from a real generation state, capturing one full model decode forward in a CUDA graph, and timing repeated graph replays at fixed effective batch size B (i.e. number of queries) and context sizes (hence, same total memory usage and total KV cache size, see Fig. 5). In practice, in Tab. 6 we find no significant changes in inference time with respect to the standard model when cross-attending between multiple lanes within a batch.

Finally, we note that, while these observations might not hold for a much larger number of lanes $N \gg 4$, a small number $N \leq 8$ is typically sufficient and larger batch sizes can still benefit by mixing multi-lanes attention with independent sampling strategies as shown in Fig. 4.

In particular, *this analysis is not valid during training time*, where the model's forward is not typically memory-bound. In this case, the additional FLOPs introduced by LaneRoPE can have some impact

[2]Whole-model FLOPs are the standard forward estimate plus attention, $\texttt{Whole-model FLOPs} = 2 \cdot N_{\text{params}} \cdot B + \texttt{Attention FLOPs}$ with $B = 8$. Attention FLOPs are estimated as $\texttt{Attention FLOPs} = \text{n_layers} \cdot 4 \cdot (N \cdot \text{ctx}) \cdot \text{head_dim} \cdot \text{n_heads} \cdot (B/N) \cdot \#\,\text{queries} \propto \#\,\text{queries} = N$.

DeepSeek-R1-Distill-Qwen-1.5B					
Lanes N	**@256**	**@2304**	**@4352**	**@6400**	**@8448**
GPU time for full-model inference (ms per token)					
1 / base	5.960	6.291	6.554	6.788	7.081
2	5.949	6.303	6.564	6.808	7.111
4	5.959	6.427	6.672	6.927	7.184
GPU time in the attention kernel (ms per token)					
1 / base	0.370	0.701	0.990	1.214	1.517
2	0.366	0.710	0.991	1.231	1.534
4	0.376	0.777	1.049	1.296	1.553
DeepSeek-R1-Distill-Qwen-7B					
Lanes N	**@256**	**@2304**	**@4352**	**@6400**	**@8448**
GPU time for full-model inference (ms per token)					
1 / base	13.646	14.222	14.704	15.281	15.778
2	13.639	14.245	14.743	15.270	15.847
4	13.663	14.345	14.927	15.418	15.923
GPU time in the attention kernel (ms per token)					
1 / base	0.421	1.013	1.501	2.066	2.477
2	0.414	1.023	1.529	1.968	2.461
4	0.401	1.074	1.633	2.032	2.429

Table 6: GPU time (*in milliseconds*) measured with `DeepSeek-R1-Distill-Qwen-1.5B` and `DeepSeek-R1-Distill-Qwen-7B` at *total* batch size $B = 8$ on an A100 GPU, with flashInfer backend and `bfloat16`. $N \geq 1$ uses multi-lane attention within groups of N out of B sequences. The first half of the table measures pure GPU time (via CUDA graph) of the whole model's forward pass to generate the next token (i.e. $B = 8$ total tokens) at different context lengths. The second half profiles the time spent in the attention kernel in the same forward.

on the training cost. On the other hand, Tab. 5 shows that the additional FLOPs represent only a fraction of the total cost (even on context lenght much longer than those we considered in this work), indicating the training overhead is bounded. Morever, and most importantly, LaneRoPE can be initialized with pre-trained standard transformers and just fine-tuned with cross-lane attention enabled. Hence, we argue this additional cost plays a minor role during training.

G Synthetize Collaborative Reasoning Traces

Below are the system prompt used to instruct the model on how the conversation will work and the user prompt used to communicate to the assistant the query and the text generated by the other assistants so far.

At each step, we prompt the model from scratch, pre-filling its own message with the text it generated in previous iterations and adding the text generated by the other assistants in the user prompt. An assistant's message at each step ends after 128 new tokens and the assistants generate for up to 30 steps. Note also that, since the user message is constantly updated, there is no reuse of the Key-Value cache in consecutive messages. In total, we generate 3 conversations with $N = 2, 3$ or 4 assistants for a subset of questions of *DeepScaleR-Preview-Dataset* (Luo et al., 2025)., resulting in about 50k raw conversations (pre-filtering). In the final data filtration phase (after filtering out any conversation with wrong answers), we score each conversation by counting the number of occurrences of other assistants' names in an assistant's completion, together with few other key phrases that suggest some disagreement happened (e.g. "Alice you are wrong"), and keep those with a count above $2N$.

System prompt

```
You are one of {N} helpful assistants (namely {assistants_list})
    collaborating to solve a problem together, while writing their
    thoughts in parallel.

Assistants collaborate without redundant work.
Each assistant's thoughts (i.e. what they write within <think> and </
    think> tags) are visible to all other assistants while they are
    being written.
In other words, as each assistant writes its thoughts, it can see all
    the thoughts which are simultaneously being written by all other
    assistants as reported in the last user message.
To be precise, you will see other assistant's current thoughts in
    previous messages under a "### <assistant-name>'s current thoughts
    ###" header.
Finally, the user will communicate you your name and, then, you will
    write your own thoughts.

You will take into account what other assistants are doing. If another
    assistant gives you suggestions, you should address them.
To keep the communication efficient, decompose long thoughts into short
     and atomic thoughts in new lines.
This is also enforced by interrupting lines that are too long and
    whenever a new sentence is started (hence, do not use the full stop
     "." unless you are done with your thought).

Every sentence and line within the <think> and </think> tags should
    focus on reaching the solution collaboratively as efficiently as
    possible, since each assistant can only write 40 lines.
You should collaborate with each other by following diverse solution
    strategies, doing different parts of the problem, double-checking
    each other's results, trying different approaches, or any other
    means.
If you realize you are currently doing the same thing that another
    assistant has already done or is in process of doing, you
    acknowledge it and stop (e.g. Alice may say 'Wait, I was doing the
    same as Bob...') and change to a different task right away, so as
    to avoid doing redundant work.
You should also use the visible thoughts within the <think> tags to
    decide how to best collaborate without doing the same work twice.
You should periodically check what other assistants are doing and
    adjust your actions accordingly in order to collaborate as
    efficiently as possible.

Finally, after the thinking process is concluded with the </think> tag,
     you are expected to give the final answer very quickly, i.e.
    within few tokens.
Hence, do NOT close the thinking process with the tag </think> before
    you have completed the collaboration and reached an agreement on
    the final answer with the other assistants.

It is extremely important you avoid doing the same work twice to make
    best use of the limited thinking budget available, so communicate
    well and efficiently!

From now on, you are {assistant_name}.
Now solve the next problem together. Keep track of who does what work
    and communicate to avoid doing the same work twice.
```

User message

```
{QUERY}

Here are the current thoughts of the other assistant

### {agent_name}'s current thoughts ###

{other-assistant-thoughts}

### End of {agent_name}'s current thoughts ###


From now on, you are {agent_name}.

Don't forget to look at what the other assistants are doing. Coordinate
     with them and avoid doing the same work twice to solve the problem
     efficiently! If you have reached a conclusion and you know the
    answer, interrupt the thinking process with </think>. Only then,
    provide the final answer within the <answer> and </answer> tags.
```

G.1 SFT EXAMPLES

The following text boxes show a single SFT sample with two lanes, one for Alice and one for Bob. Note that we have removed or replaced several unicode characters of the chat template for Latex compatibility.

Alice's example

```
<|beginofsentence|><|User|>
You are one of 2 helpful assistants (namely Alice and Bob)
    collaborating to solve a problem together, while writing their
    thoughts in parallel.

Assistants collaborate without redundant work.
Each assistant's past and current thoughts are visible to all other
    assistants while they are being written.
In other words, as each assistant writes its thoughts, it can see all
    the thoughts which are simultaneously being written by all other
    assistants.

You will see the other assistants unfinished steps. They will write
    those steps concurrently with you. You will take into account what
    they are doing. If another assistant gives you suggestions, you
    should address them.
Other assistants will continue writing their thoughts in the background
     while you will continue writing your own.
Since you and others both write your thoughts in parallel, you will
    initially see only partial (unfinished) thoughts that others will
    continue in parallel, while you write yours.
Other assistants may write new thoughts while you are writing yours.
You will use these partial thoughts to decide how best to collaborate
    without doing the same work twice. You will periodically check what
     other assistants are doing and you should adjust your actions
    based on what they are doing so you collaborate efficiently with
    them.
If what you are currently doing is the same thing that another
    assistant has already done or is in process of doing, you will stop
     (e.g. Alice may say Wait, I was doing the same as Bob ...) and
    change to a different task right away, so as to avoid doing
    redundant work.
```

```
Now solve the next problem together. Keep track of who does what work
    and communicate well to avoid doing the same work twice.

From now on, you act as Alice.

A cell phone plan costs $20$ dollars each month, plus $5$ cents per
    text message sent, plus $10$ cents for each minute used over $30$
    hours. In January Michelle sent $100$ text messages and talked for
    $30.5$ hours. How much did she have to pay?
Let's think step by step and output the final answer within \boxed{}.
<|Assistant|>
<think>
Okay, let's tackle this problem step by step.
First, the base cost is $20 per month.
Now, Michelle sent 100 text messages.
Each text is 5 cents, so 100 * $0.05.
Wait, Bob already mentioned the text messages.
Let me check the time part.
She talked for 30.5 hours.
The plan covers 30 hours, so the overage is 0.5 hours.
0.5 hours is 30 minutes.
Each minute over is 10 cents, so 30 * $0.10.
Bob is handling the time part, so I shouldn't repeat that.
Wait, the total would be base + text cost + overage.
Base: $20.
Texts: $5 (100*0.05).
Overage: $3 (30*0.10).
Total: 20+5+3=28.
Bob said 28, so that's consistent.
I need to make sure there's no miscalculation here.
Wait, 0.5 hours is 30 minutes, yes.
30 minutes * $0.10 = $3.
Texts: 100 * 0.05 = 5.
All adds up to $28.
No errors here.
So the answer is $28.
Since Bob already confirmed this, I agree with his calculation.
No need to redo anything.
Confirming once more to be safe.
Yes, all parts accounted for.
Final answer should be 28.
</think>
\boxed{28}<|endofsentence|>
```

Bob's example

```
<|beginofsentence|><|User|>
You are one of 2 helpful assistants (namely Alice and Bob)
    collaborating to solve a problem together, while writing their
    thoughts in parallel.

Assistants collaborate without redundant work.
Each assistant's past and current thoughts are visible to all other
    assistants while they are being written.
In other words, as each assistant writes its thoughts, it can see all
    the thoughts which are simultaneously being written by all other
    assistants.

You will see the other assistants unfinished steps. They will write
    those steps concurrently with you. You will take into account what
```

```
    they are doing. If another assistant gives you suggestions, you
    should address them.
Other assistants will continue writing their thoughts in the background
     while you will continue writing your own.
Since you and others both write your thoughts in parallel, you will
    initially see only partial (unfinished) thoughts that others will
    continue in parallel, while you write yours.
Other assistants may write new thoughts while you are writing yours.
You will use these partial thoughts to decide how best to collaborate
    without doing the same work twice. You will periodically check what
     other assistants are doing and you should adjust your actions
    based on what they are doing so you collaborate efficiently with
    them.
If what you are currently doing is the same thing that another
    assistant has already done or is in process of doing, you will stop
     (e.g. Alice may say Wait, I was doing the same as Bob ...) and
    change to a different task right away, so as to avoid doing
    redundant work.

Now solve the next problem together. Keep track of who does what work
    and communicate well to avoid doing the same work twice.

From now on, you act as Bob.

A cell phone plan costs $20$ dollars each month, plus $5$ cents per
    text message sent, plus $10$ cents for each minute used over $30$
    hours. In January Michelle sent $100$ text messages and talked for
    $30.5$ hours. How much did she have to pay?
Let's think step by step and output the final answer within \boxed{}.
<|Assistant|>
<think>
Okay, let's tackle this problem step by step.
First, the base cost is $20 per month.
Now, Michelle sent 100 text messages.
The cost per text is 5 cents, so that's 100 * $0.05.
Wait, Alice already calculated the text messages part.
Let me check the time overage.
She talked for 30.5 hours, so over the 30-hour limit by 0.5 hours.
Convert 0.5 hours to minutes: 0.5 * 60 = 30 minutes.
Each minute over is 10 cents, so 30 * $0.10.
Alice mentioned the time part, so I don't need to repeat that.
Let me just confirm the calculations.
Texts: 100 * 0.05 = $5.
Time overage: 30 * 0.10 = $3.
Total cost: 20 + 5 + 3 = $28.
Wait, Alice already did the text cost.
She said texts are $5 and overage $3, so total $28.
I should check if my numbers match.
Yes, 100 text messages at 5 cents is $5.
0.5 hours over is 30 minutes, 30*0.10 is $3.
All adds up.
No mistakes here.
So the final answer should be $28.
I should confirm no other parts are missed.
The problem didn't mention other fees, so yes, total is 20 +5+3=28.
Looks correct.
Since Alice already worked through it and I'm just verifying, I can
    agree.
No need to re-explain, just confirm the answer.
All right, so the answer is 28.
</think>
\boxed{28}<|endofsentence|>
```