Continuous Chain of Thought Enables Parallel Exploration and Reasoning

Halil Alperen Gozeten [*1] **M. Emrullah Ildiz** [*1] **Xuechen Zhang** [1] **Hrayr Harutyunyan** [2] **Ankit Singh Rawat** [2] **Samet Oymak** [1]

Abstract

We propose CoT2, a framework using continuously-valued tokens that enables language models to track multiple reasoning paths in parallel and provide a novel CoT2 supervision strategy where we match the softmax outputs to the empirical token distributions of a set of target traces. Theoretically, we show that CoT2 offers sample-complexity benefits and construct a one-layer transformer that solves the subset-sum problem with sufficient embedding capacity. We also introduce continuous sampling methods, showing that reinforcement learning with CoT2 notably improves logical reasoning performance compared to discrete and continuous baselines.

1. Introduction

Chain-of-thought (CoT) strategies (Wei et al., 2022), when paired with strong base models, have achieved immense success and facilitated progress in remarkably challenging tasks, such as solving AIME or IOI problems (Guo et al., 2025; Jaech et al., 2024). In essence, CoT boosts the expressive capability of the base model through autoregressive generation, a principle that also underlies the recent efforts on test-time compute scaling (Snell et al., 2024). Despite these advances, modern language model architectures may not fully utilize their potential for a few reasons. First is their discrete sampling of tokens—selecting a single token at each decoding step from a vocabulary of v tokens. This limits the model to emitting at most $\log_2(v)$ bits per sample, or more specifically, the Shannon entropy of the softmax output. This contrasts with the $O(d)$ bits each token embedding can store, where d is the embedding dimension. Secondly, discrete sampling can cause the model to *commit* to certain solutions and avoid exploring alternatives (Yao et al., 2023). A practical method to address this is sampling multiple CoT traces and aggregating them, either through consistency (Wang et al., 2022) or best-of-N decoding (Ouyang et al., 2022) through more test-time computation.

In this work, we propose *CoT with Continuous Tokens* (CoT2) to address these issues, building on COCONUT (Hao et al., 2024). Instead of sampling a single token, CoT2 samples or deterministically selects a continuous superposition of tokens according to the softmax distribution. This enables models to embed more information per token and simultaneously track multiple reasoning paths, potentially emulating self-consistency or best-of-N decoding within a single trace. We provide the following technical contributions:

- **Mechanistic and theoretical study of CoT2:** We quantify CoT2's benefits by examining the *Minimum Non-Negative Sum* (MNNS) problem, a generalization of the classical Subset Sum problem, along with related tasks like ProntoQA (Saparov & He, 2022), which inherently benefit from parallel search. We demonstrate that a single-layer transformer can solve MNNS using CoT2, indicating transformers' ability to track multiple reasoning paths. Under a trajectory decoupling assumption, we theoretically study CoT2 decoding methods:
  - **Base CoT2:** deterministic inference creating continuous tokens from the full softmax output (Sec.2&3);
  - **CoT2-MTS (multi-token sampling):** our method sampling K discrete tokens from softmax, averaged into a continuous token (Sec.4);

  showing base CoT2 aggregates all reasoning paths, while CoT2-MTS generalizes CoT by tracking K paths, and establishing CoT2's sample complexity benefits.

[*]Equal contribution [1]University of Michigan - Ann Arbor [2]Google Research NYC. Correspondence to: Halil Alperen Gozeten <alperen@umich.edu>.

Accepted at Methods and Opportunities at Small Scale (MOSS), ICML 2025, Vancouver, Canada.

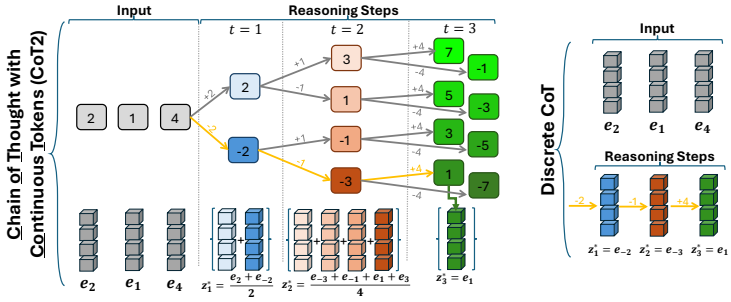


Figure 1. Illustration of CoT2 and discrete CoT for Minimum Non-Negative Sum (MNNS) task with $m = 3$. Input numbers are $2, 1, 4$, and the correct path $(-2, -3, 1)$ is highlighted with yellow arrows, corresponding to the discrete CoT supervision. CoT2 supervision for reasoning steps $t \in \{1, \ldots, m-1\}$ is the average of embeddings of reachable states, and for $t = m$ is the embedding of the answer.

- **Supervision and reinforcement for CoT2:** We introduce the continuous supervision strategy CSFT for explicitly tracking multiple teacher traces by fitting a target softmax map of token distributions within the trace. Our method highlights fundamental tradeoffs between CoT2 accuracy and embedding dimensions. We also propose reinforcement policy optimization methods for CoT2, with MTS sampling as the primary strategy to control parallelism and a purely continuous sampling method. Experiments demonstrate GRPO-based RL with CoT2 enhances accuracy beyond SFT or CSFT, indicating RL effectively prioritizes relevant reasoning paths. Related work is discussed in Appendix A.

2. Problem Setup

Assume we have an input context $\boldsymbol{X} \in \mathbb{R}^{n \times d}$, consisting of n rows of d-dimensional embeddings. The goal is to generate m tokens given $\boldsymbol{X}$, where the final token is discrete and evaluated under a performance metric (e.g., accuracy or reward). For steps 1 to $m-1$, the model outputs continuous *thought tokens* $\{z_t\}_{t \in [m-1]}$ that enable reasoning. At step m, the model outputs a discrete token z_m from a vocabulary of size v. In the rest of the paper, we investigate training strategies that enhance final performance beyond standard discrete next-token prediction

Formally, let $\boldsymbol{E} = [\boldsymbol{e}_1, \ldots, \boldsymbol{e}_v]^\top \in \mathbb{R}^{v \times d}$ be the embedding matrix corresponding to the vocabulary of v tokens, where $\boldsymbol{e}_i \in \mathbb{R}^d$ represents the embedding of the ith token. We define the next-token prediction model LM_θ parameterized by θ that assigns, at each step t, a probability distribution over possible next tokens given the prefix $z_{<t}$ and context $\boldsymbol{X}$. Concretely, for $1 \le t \le m-1$, the model outputs following probability distribution over the v vocabulary entries via a softmax operation:

$$\mathrm{LM}_\theta(\cdot \mid z_{<t}, \boldsymbol{X}) := \boldsymbol{\alpha}_t \text{ where } \boldsymbol{\alpha}_t = \left[\alpha_{t,1}, \ldots, \alpha_{t,v}\right] \in \Delta^{v-1}, \text{ i.e. } \alpha_{t,i} \ge 0 \text{ and } \sum_{i=1}^{v} \alpha_{t,i} = 1$$

We then form the continuous token as the convex combination of all tokens in the vocabulary:

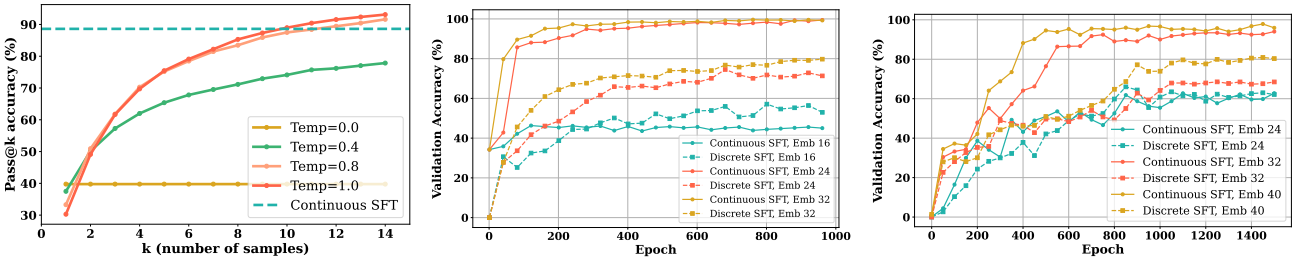
$$z_t = \boldsymbol{E}^\top \boldsymbol{\alpha}_t \in \mathbb{R}^d, \quad \forall 1 \le t \le m-1$$

Hence each continuous token z_t is a linear combination of the vocabulary embeddings. At the final step $t = m$, the model samples a discrete token $z_m \in \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_v\}$ from its policy distribution $\mathrm{LM}_\theta(\cdot \mid z_{<m}, \boldsymbol{X}) = \boldsymbol{\alpha}_m$. Finally, we note that we assume that the answer depends only on the final discrete token z_m merely for simplicity; the same framework naturally extends to decoding multiple final discrete tokens after continuous ones. We refer to this decoding strategy as **base CoT2** and observe that it results in a deterministic reasoning chain because the continuous tokens are precisely determined by the softmax map. In Section 4, we will introduce stochastic alternatives, such as **CoT2-MTS**, to facilitate generative reasoning.

3. CSFT: A Supervised Training Method for CoT2

In this section, we present our method of continuous supervised training to learn intermediate thought tokens as "soft" targets rather than "hard" target tokens, as described in Section 2. Specifically, we provide the model with convex combinations of vocabulary embeddings, which allows the model flexibility in those reasoning steps. Such an approach is particularly suitable when the task accuracy depends only on the final token or token distribution. Formally, at each reasoning step $t = 1, \ldots, m-1$, the supervision specifies a target probability distribution

$$\boldsymbol{\alpha}_t^* = \left[\alpha_{t,1}^*, \ldots, \alpha_{t,v}^*\right] \in \Delta^{v-1},$$



(a) Continuous vs. discrete SFT (Pass@k) accuracies for different temperatures on MNNS task.

(b) Training performance vs. embedding dimension for continuous and discrete SFT on MNNS task.

(c) Training performance vs. embedding dimension for continuous and discrete models on ProsQA.

Figure 2. The teacher distribution for the continuous model is derived from a search algorithm. **(a):** The figure illustrates that the discrete model requires multiple samplings (Pass@k) to match the single-sample performance of the continuous model on MNNS (10-run average). **Setting:** 4 input digits in $1-9$; 1-layer, 1-head GPT2 with $d = 24$. **(b-c):** The figures reveal that above a certain embedding dimension threshold, the continuous model is superior to discrete in tasks involving search, like MNNS and ProsQA. **Setting (b):** 4 input digits in $1-9$; 2-layer, 2-head GPT2 with $d \in \{16, 24, 32\}$. **(c)**: 4-layer, 4-head GPT2 with $d \in \{24, 32, 40\}$.

where $\alpha^*_{t,i} \geq 0$ and $\sum_{i=1}^{v} \alpha^*_{t,i} = 1$. We train the model to align its predicted distribution $\boldsymbol{\alpha}_t$ to the supervision distribution $\boldsymbol{\alpha}^*_t$ rather than one-hot labels, with the help of a divergence-based loss:

$$\mathcal{L}_{\text{cont}}(\theta; \boldsymbol{X}, t) = D\left(\boldsymbol{\alpha}^*_t \,\middle\|\, \boldsymbol{\alpha}_t\right),$$

where $D(\cdot\|\cdot)$ is the cross-entropy (or equivalently KL divergence) between two distributions. This approach can also be viewed as *token-level knowledge-distillation*, where the teacher distribution $\boldsymbol{\alpha}^*_t$ may be obtained through a logic/search algorithm. At the final step $t = m$, we typically have a discrete target $\boldsymbol{z}^*_m \in \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_v\}$, so that $\boldsymbol{\alpha}^*_m$ is one-hot distribution placing probability 1 on that target token and 0 elsewhere. This is equivalent to employing a standard cross-entropy loss $-\log \text{LM}_\theta(\boldsymbol{z}^*_m \mid \boldsymbol{z}_{<m}, \boldsymbol{X})$ at the final step. Hence, for each training example, the total loss for the proposed continuous supervised training is the sum of the continuous-token divergence losses:

$$\mathcal{L}_{\text{CSFT}}(\theta; \boldsymbol{X}) = \sum_{t=1}^{m} \mathcal{L}_{\text{cont}}(\theta; \boldsymbol{X}, t) \tag{1}$$

By minimizing $\mathcal{L}_{\text{CSFT}}(\theta)$, we teach the model to learn the soft targets $\boldsymbol{\alpha}^*_t$ at each step and to predict the correct final discrete token. In the above training procedure, inspired by the discussions in Bachmann & Nagarajan (2024); Bengio et al. (2015), we consider two ways of providing prefixes to the language model:

1. **Teacher forcing:** Each step t is conditioned on the prefix $\boldsymbol{z}^*_{<t}$, providing the model access to previous ground-truth tokens during prediction. Specifically, for $t' < t$, the input $\boldsymbol{z}^*_{t'} = \boldsymbol{E}^\top \boldsymbol{\alpha}^*_{t'}$ is a convex combination of vocabulary tokens.
2. **Self-feeding:** At each step t, the model autoregressively uses its generated outputs $\boldsymbol{z}_{<t}$ during training. Specifically, the continuous token $\boldsymbol{z}_t = \boldsymbol{E}^\top \boldsymbol{\alpha}_t$, a convex combination of vocabulary embeddings, is fed back as part of the prefix.

It is also worth noting that one may apply *temperature scaling* or *thresholding* to $\boldsymbol{\alpha}_t$ before forming $\boldsymbol{z}_t$ in order to filter the model's predictions. In our experiments, we find that teacher forcing leads to superior performance for CSFT, even though at inference time, the model runs in an autoregressive manner, as discussed below. See Appendix E for further discussion.

Inference. At inference time, the model does not use ground-truth distributions $\boldsymbol{\alpha}^*_t$. Instead, at each continuous step $t < m$, the model autoregressively generates a continuous token $\boldsymbol{z}_t = \boldsymbol{E}^\top \boldsymbol{\alpha}_t$ from its own distribution $\boldsymbol{\alpha}_t$ to form the prefix for the next step. At the final step, it samples a discrete token from $\boldsymbol{\alpha}_m = \text{LM}_\theta(\cdot \mid \boldsymbol{z}_{<m}, \boldsymbol{X})$.

Discrete baseline. We use teacher-forced training, predicting the next token conditioned on ground-truth previous tokens using standard cross-entropy loss. This baseline enforces each $\boldsymbol{z}_t$ as a token from vocabulary $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_v\}$, making it a special case of CSFT with one-hot vectors $\boldsymbol{\alpha}_t \in \Delta^{v-1}$. The model minimizes the following summed objective:

$$\mathcal{L}_{\text{SFT}}(\theta; \boldsymbol{X}) = \sum_{t=1}^{m} -\log \text{LM}_\theta\left(\boldsymbol{z}^*_t \mid \boldsymbol{z}^*_{<t}, \boldsymbol{X}\right). \tag{2}$$

3.1. Tasks Requiring Exploration over States

In this subsection, we illustrate the CSFT training described in (1) on tasks that require *exploration* over multiple states. Consider that the vocabulary is sufficiently large that each state g of the task can be assigned a unique embedding. Then, we let Γ_t be the set of all states that could result from building upon step $(t-1)$, where $\Gamma_0 = \{g_0\}$ for the initial state g_0. For each

element $g \in \Gamma_t$, we assign a probability $\alpha^*_{t,g}$ that reflects how many times that state occurs under a search process. Then, $\boldsymbol{\alpha}^*_t$ is formed by normalizing these probabilities into a distribution on Γ_t:

$$\alpha^*_{t,g} = \frac{\text{count}_t(g)}{\sum_{h \in \Gamma_t} \text{count}_t(h)}, \tag{3}$$

where $\text{count}_t(g)$ is the number of times state g appears among all expansions at step t. At the final step $t = m$, we select exactly one correct state from Γ_m, so that $\boldsymbol{\alpha}^*_m$ is a one-hot vector:

$$\alpha^*_{m,g} = \begin{cases} 1, & \text{if } g \text{ is the correct final state,} \\ 0, & \text{otherwise.} \end{cases}$$

Specifically, we apply this framework to three illustrative tasks. First, we devise a search task, **MNNS (Minimum Non-Negative Sum)**, which requires assigning plus or minus signs to a given set of numbers to achieve the smallest possible sum that remains non-negative. This task can be considered as traversing a binary graph, where each step corresponds to selecting either a positive or negative sign, resulting in multiple potential paths representing different sums. Second, **ProntoQA** involves logical reasoning within graphs composed of words as nodes and directed hops as edges between them, where the aim is to determine whether a specific target node is reachable from a starting node within a fixed number of traversal steps. Finally, **ProsQA** similarly assess which of two candidate nodes is reachable from a common starting node, again within a predetermined number of traversal steps. More details on tasks are provided in Appendix B.

3.2. Results and Discussion

In our experiments on MNNS, ProsQA, and ProntoQA tasks, CSFT significantly outperforms discrete CoT baseline once the embedding dimension surpasses a moderate threshold (Figures 2b, 2c), and also enables faster convergence. Continuous tokens represent multiple partial expansions in parallel, providing a 'search-like' capability that yields higher accuracy. As shown in Figure 2a, the discrete model needs multiple samplings (pass@k) to match the single-attempt performance of CoT2, highlighting continuous tokens' ability to avoid early accumulating errors, aligning with "snowballing errors" discussed in (Bachmann & Nagarajan, 2024). Moreover, though the continuous approach needs more embedding capacity to allow its distributional representations at each step, it can then achieve strong performance with fewer layers and heads compared to the discrete model, as further demonstrated by the results in Appendix E.1. We also provide experiments on ProntoQA in Appendix E.1, which confirm similar findings to results on ProsQA.

4. Reinforcement Learning Methods for CoT2

In this section, we describe how to apply RL with continuous output tokens. Specifically, we explore GRPO training on top of continuous or discrete models that are supervised trained based on Section 3 for the MNNS, ProntoQA, and ProsQA tasks. By illustrating two sampling methods for GRPO, we demonstrate how a model trained with discrete SFT can be adapted to produce continuous outputs. We assume a sparse reward setting where the reward is 1 for a correct final answer and 0 otherwise. For further details on this method and Dirichlet sampling, we refer readers to Appendix C.

4.1. Multi-Token Sampling

We emulate the rollout of a continuous token by sampling a fixed number of $K \leq v$ discrete tokens and averaging them at steps $t = 1, \ldots, m-1$. We refer to this hybrid method as *CoT2 with MTS* (multi-token sampling). For the GRPO objective, we propose the following method to calculate the policy ratio for continuous tokens. Specifically, assume at step t we sample discrete tokens $\boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_K}$ with probabilities $\alpha_{t,i_1}, \ldots, \alpha_{t,i_K}$ under the current policy and probabilities $\alpha^{\text{old}}_{t,i_1}, \ldots, \alpha^{\text{old}}_{t,i_K}$ under the old policy. We define the policy ratio for these continuous steps by dividing geometric means:

$$r_t(\theta) = \frac{\text{LM}_\theta(z_t \mid z_{<t}, \boldsymbol{X})}{\text{LM}_{\theta_{\text{old}}}(z_t \mid z_{<t}, \boldsymbol{X})} = \left(\frac{\alpha_{t,i_1} \cdots \alpha_{t,i_K}}{\alpha^{\text{old}}_{t,i_1} \cdots \alpha^{\text{old}}_{t,i_K}} \right)^{1/K} \quad \text{for } t = 1, \ldots, m-1.$$

The geometric mean ensures that the ratio for each continuous step remains on the same scale as the final discrete token's ratio and, thus, helps avoid overly large or small updates in the GRPO objective and provides more stable training compared to the direct multiplication of probabilities. Once this ratio is computed, we then average the K sampled tokens to form z_t, which is fed to the model as the query for the next prediction step. At the final step $t = m$, where the token $z_m = \boldsymbol{e}_j$ is

Table 1. Validation accuracy and token-level entropy of CoT2-MTS sampling GRPO on the discrete model under different rollout sizes K for MNNS task. We use 4 input digits in 1-9 with 1-layer, 1-head GPT2 at embedding dimensions 24 and 32, with SFT accuracies of 39.76% and 43.50%, respectively.

	K	Val. Accuracy (%)		Val. Entropy (SFT → SFT+GRPO)			
		SFT	SFT+GRPO	token_1	token_2	token_3	token_4
24	1	39.76	49.01	0.3218 → 0.0314	0.5858 → 0.0712	0.5499 → 0.1576	0.4786 → 0.1718
	3		52.60	0.3717 → 0.0647	0.7461 → 0.2120	0.8006 → 0.3312	0.5338 → 0.1529
	6		49.69	0.4524 → 0.1242	0.7738 → 0.3361	0.8364 → 0.6615	0.5134 → 0.2159
32	1	43.50	51.61	0.3618 → 0.0143	0.6331 → 0.0395	0.3518 → 0.2631	0.1962 → 0.1182
	3		55.66	0.3886 → 0.0412	0.7101 → 0.0904	0.5447 → 0.5757	0.2863 → 0.1734
	6		50.38	0.4224 → 0.0632	0.7915 → 0.2161	0.6077 → 0.8481	0.2780 → 0.1514

discrete with $j \in [v]$ denoting its index, the policy ratio is simply the probability ratio of selecting that token:

$$r_m(\theta) = \frac{\mathrm{LM}_\theta\left(z_m \mid z_{<m}, \boldsymbol{X}\right)}{\mathrm{LM}_{\theta_{\text{old}}}\left(z_m \mid z_{<m}, \boldsymbol{X}\right)} = \frac{\alpha_{m,j}}{\alpha_{m,j}^{\text{old}}}.$$

4.2. Dirichlet Sampling

In this section, we present another method for generating continuous tokens at each step by interpreting the model's output distribution $\boldsymbol{\alpha}_t \in \Delta^{v-1}$ as concentration parameters of a Dirichlet distribution over the $v-1$ simplex. We introduce a scaling hyperparameter $\gamma > 0$ and define the Dirichlet distribution with the parameters $\gamma\,\boldsymbol{\alpha}_t = (\gamma\alpha_{t,1}, \ldots, \gamma\alpha_{t,v})$. Without this scaling, directly using $\boldsymbol{\alpha}_t$ as parameters often causes training instability, particularly when many $\alpha_{t,i}$ values are small. We then sample a point $\hat{\boldsymbol{\alpha}}_t \in \Delta^{v-1}$ from the resulting distribution $\mathrm{Dir}\left(\boldsymbol{\alpha}_t\right)$. After sampling, we form the continuous token by mapping $\boldsymbol{z}_t = \boldsymbol{E}^\top \hat{\boldsymbol{\alpha}}_t \in \mathbb{R}^d$, which becomes the query for the next step. We denote the Dirichlet densities induced by current and old policies as $f_\theta(\boldsymbol{z}_t)$ and $f_{\theta_{\text{old}}}(\boldsymbol{z}_t)$, respectively. Accordingly, we define the policy ratio at a continuous step $t < m$ as:

$$r_t(\theta) = \frac{\mathrm{LM}_\theta(z_t \mid z_{<t}, \boldsymbol{X})}{\mathrm{LM}_{\theta_{\text{old}}}(z_t \mid z_{<t}, \boldsymbol{X})} = \frac{f_\theta(\boldsymbol{z}_t)}{f_{\theta_{\text{old}}}(\boldsymbol{z}_t)},$$

The above definition parallels how we compute probability ratios for discrete actions, but replaces the categorical pmf with a continuous Dirichlet pdf. At the final step $t = m$, we sample a discrete token $\boldsymbol{z}_m \in \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_v\}$ from $\boldsymbol{\alpha}_m$, and use the standard policy ratio given by (5). At inference, we follow the autoregressive procedure in Section 3 by creating a convex combination of vocabulary tokens.

4.3. Results and Discussion of Policy Optimization for CoT2

MNNS evaluation: Table 1 provides our results for the MNNS task and demonstrates that, for each $K \in \{1, 3, 6\}$, CoT2-MTS significantly improves validation accuracy relative to the discrete SFT baseline (39.76%), with moderate K yielding the best final performance. We also observe that smaller K-values correspond to larger reductions in token-level entropies, suggesting that the model becomes more confident at each intermediate step by learning to commit to fewer tokens. This suggests a curriculum on K—starting small and gradually increasing—could potentially further improve the training on the MNNS task. Interestingly, the third token's entropy remains relatively high, which might indicate that the model continues to hedge among several partial expansions at that step, which may help preserve useful diversity of reasoning. Therefore, CoT2-MTS enables a model trained with discrete SFT to produce continuous outputs and helps it achieve better performance. Finally, Appendix E.2 shows that the CoT2 model with CSFT achieves a strong performance once the embedding dimension is sufficiently large (compared to the results in Table 1), however, it can be further improved with GRPO with Dirichlet Sampling.

5. Theoretical Analysis

In this section, we first present the construction of a single-layer transformer that solves the MNNS task using an attention layer followed by a mixture-of-experts MLP layer. We then provide a theoretical comparison between the sample-complexity advantages of CoT2 strategies over discrete CoT, with details provided in Appendix D.

References

Aggarwal, P. and Welleck, S. L1: Controlling how long a reasoning model thinks with reinforcement learning. *arXiv preprint arXiv:2503.04697*, 2025.

Bachmann, G. and Nagarajan, V. The pitfalls of next-token prediction. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=76zq8Wkl6Z.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks, 2015. URL https://arxiv.org/abs/1506.03099.

Blei, D. M., Ng, A. Y., and Jordan, M. I. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022, March 2003. ISSN 1532-4435.

Cheng, J. and Van Durme, B. Compressed chain of thought: Efficient reasoning through dense representations. *arXiv preprint arXiv:2412.13171*, 2024.

Deng, Y., Prasad, K., Fernandez, R., Smolensky, P., Chaudhary, V., and Shieber, S. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

Deng, Y., Choi, Y., and Shieber, S. From explicit cot to implicit cot: Learning to internalize cot step by step. *arXiv preprint arXiv:2405.14838*, 2024.

Dziri, N., Lu, X., Sclar, M., Li, X. L., Jiang, L., Lin, B. Y., West, P., Bhagavatula, C., Bras, R. L., Hwang, J. D., Sanyal, S., Welleck, S., Ren, X., Ettinger, A., Harchaoui, Z., and Choi, Y. Faith and fate: Limits of transformers on compositionality, 2023. URL https://arxiv.org/abs/2305.18654.

Feng, G., Zhang, B., Gu, Y., Ye, H., He, D., and Wang, L. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=qHrADgAdYu.

Geiping, J., McLeish, S., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., Kailkhura, B., Bhatele, A., and Goldstein, T. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

Giannou, A., Rajput, S., Sohn, J.-Y., Lee, K., Lee, J. D., and Papailiopoulos, D. Looped transformers as programmable computers. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J. (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 11398–11442. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/giannou23a.html.

Gloeckle, F., Idrissi, B. Y., Rozière, B., Lopez-Paz, D., and Synnaeve, G. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Goyal, S., Ji, Z., Rawat, A. S., Menon, A. K., Kumar, S., and Nagarajan, V. Think before you speak: Training language models with pause tokens. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ph04CRkPdC.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Hao, S., Sukhbaatar, S., Su, D., Li, X., Hu, Z., Weston, J., and Tian, Y. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Ildiz, M. E., Huang, Y., Li, Y., Rawat, A. S., and Oymak, S. From self-attention to markov models: Unveiling the dynamics of generative transformers, 2024. URL https://arxiv.org/abs/2402.13512.

Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Kamath, S., Orlitsky, A., Pichapati, D., and Suresh, A. T. On learning distributions from their samples. In Grünwald, P., Hazan, E., and Kale, S. (eds.), *Proceedings of The 28th Conference on Learning Theory*, volume 40 of *Proceedings of Machine Learning Research*, pp. 1066–1100, Paris, France, 03–06 Jul 2015. PMLR. URL https://proceedings.mlr.press/v40/Kamath15.html.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Merrill, W. and Sabharwal, A. The expressive power of transformers with chain of thought. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NjNGlPh8Wh.

Nanda, N., Chan, L., Lieberum, T., Smith, J., and Steinhardt, J. Progress measures for grokking via mechanistic interpretability, 2023. URL https://arxiv.org/abs/2301.05217.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., et al. Show your work: Scratchpads for intermediate computation with language models. 2021.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Pfau, J., Merrill, W., and Bowman, S. R. Let's think dot by dot: Hidden computation in transformer language models. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=NikbrdtYvG.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Technical Report*, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Saparov, A. and He, H. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. *arXiv preprint arXiv:2210.01240*, 2022.

Saunshi, N., Dikkala, N., Li, Z., Kumar, S., and Reddi, S. J. Reasoning with latent thoughts: On the power of looped transformers. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=din0lGfZFd.

Shalev, Y., Feder, A., and Goldstein, A. Distributional reasoning in llms: Parallel reasoning processes in multi-hop reasoning. *arXiv preprint arXiv:2406.13858*, 2024.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Shazeer, N. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Shen, Z., Yan, H., Zhang, L., Hu, Z., Du, Y., and He, Y. Codi: Compressing chain-of-thought into continuous space via self-distillation. *arXiv preprint arXiv:2502.21074*, 2025.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., and et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, Oct 2017. doi: 10.1038/nature24270.

Snell, C., Lee, J., Xu, K., and Kumar, A. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Stiennon, N., Ouyang, L., Wu, J., Ziegler, D. M., Lowe, R., Voss, C., Radford, A., Amodei, D., and Christiano, P. Learning to summarize from human feedback, 2022. URL https://arxiv.org/abs/2009.01325.

Sui, Y., Chuang, Y.-N., Wang, G., Zhang, J., Zhang, T., Yuan, J., Liu, H., Wen, A., Zhong, S., Chen, H., et al. Stop overthinking: A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*, 2025.

Suzgun, M., Scales, N., Schärli, N., Gehrmann, S., Tay, Y., Chung, H. W., Chowdhery, A., Le, Q., Chi, E., Zhou, D., and Wei, J. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.824. URL https://aclanthology.org/2023.findings-acl.824/.

Thomm, J., Camposampiero, G., Terzic, A., Hersche, M., Schölkopf, B., and Rahimi, A. Limits of transformer language models on learning to compose algorithms, 2024. URL https://arxiv.org/abs/2402.05785.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Xiong, Z., Cai, Z., Cooper, J., Ge, A., Papageorgiou, V., Sifakis, Z., Giannou, A., Lin, Z., Yang, L., Agarwal, S., Chrysos, G. G., Oymak, S., Lee, K., and Papailiopoulos, D. Everything everywhere all at once: Llms can in-context learn multiple tasks in superposition, 2024. URL https://arxiv.org/abs/2410.05603.

Yang, S., Gribovskaya, E., Kassner, N., Geva, M., and Riedel, S. Do large language models latently perform multi-hop reasoning? In Ku, L.-W., Martins, A., and Srikumar, V. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10210–10229, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.550. URL https://aclanthology.org/2024.acl-long.550/.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Yu, P., Xu, J., Weston, J., and Kulikov, I. Distilling system 2 into system 1. *arXiv preprint arXiv:2407.06023*, 2024.

Yu, Q., Zhang, Z., Zhu, R., Yuan, Y., Zuo, X., Yue, Y., Fan, T., Liu, G., Liu, L., Liu, X., Lin, H., Lin, Z., Ma, B., Sheng, G., Tong, Y., Zhang, C., Zhang, M., Zhang, W., Zhu, H., Zhu, J., Chen, J., Chen, J., Wang, C., Yu, H., Dai, W., Song, Y., Wei, X., Zhou, H., Liu, J., Ma, W.-Y., Zhang, Y.-Q., Yan, L., Qiao, M., Wu, Y., and Wang, M. Dapo: An open-source llm reinforcement learning system at scale, 2025. URL https://arxiv.org/abs/2503.14476.

Zhang, X., Huang, Z., Ni, C., Xiong, Z., Chen, J., and Oymak, S. Making small language models efficient reasoners: Intervention, supervision, reinforcement. *arXiv preprint arXiv:2505.07961*, 2025.

APPENDIX

We discuss additional related work in Appendix A. We provide further implementation details in Appendix G, including those for the MNNS task (Appendix G.1), the ProntoQA/ProsQA datasets (Appendix G.2), and GRPO training (Appendix G.3). We present additional experimental results in Appendix E, and we offer details on continuous supervised training and GRPO in Appendix E.1 and Appendix E.2, respectively. Finally, we include the proofs of Propositions 1, 2, and 3 in Appendix F.

A. Related Work

The efficacy of eliciting reasoning in LLMs through chain-of-thought (CoT) prompting has been well-established (Nye et al., 2021; Wei et al., 2022; Kojima et al., 2022; Suzgun et al., 2023; Guo et al., 2025). CoT prompting provides a convenient way to increase inference-time compute and computational depth, both of which have been found to be independently useful (Pfau et al., 2024; Goyal et al., 2024; Feng et al., 2023; Merrill & Sabharwal, 2024). However, the discrete nature of CoT tokens forces sequential exploration of reasoning paths, resulting in longer reasoning paths and consequently increased inference-time compute. Furthermore, restricting reasoning to natural language can be inefficient, as groups of tokens can often be more effectively represented by a single continuous token. Thus, CoT2 offers an alternative strategy for compute-efficient reasoning and complements methods that aim to shorten/control the trace length of CoT (Aggarwal & Welleck, 2025; Zhang et al., 2025; Sui et al., 2025).

One way to address these challenges is by leveraging the implicit reasoning capabilities of transformers (Yang et al., 2024; Shalev et al., 2024). Works such as (Deng et al., 2023; 2024; Yu et al., 2024) use various techniques to obtain models that can perform reasoning internally without emitting CoT tokens. Another line of work has found looped transformers to be effective on reasoning problems (Giannou et al., 2023; Geiping et al., 2025), notably being able to mimic CoT (Saunshi et al., 2025) with a sufficient number of iterations. Our work is similar to this line of work in that continuous representations are used to perform reasoning.

Our work is most related to a recent body of work introducing LLMs capable of reasoning with explicit continuous tokens decoded autoregressively. In particular, recently proposed COCONUT (Hao et al., 2024) autoregressively feeds the last token's final-layer representation as input to the next step. Given labeled CoT data, COCONUT is trained to progressively replace discrete tokens with continuous tokens (from left to right). Shen et al. (2025) propose CODI, where an LLM with continuous CoT is supervised to produce the correct answer, while also aligning its hidden representation on the last reasoning token to that of a discrete CoT model that shares the same backbone. Cheng & Van Durme (2024) propose CCOT, where an auxiliary module is first trained to decode autoregressively a compressed representation of a discrete CoT trace, and later the main LLM is fine-tuned to produce correct answers by additionally conditioning on the generated continuous tokens. While COCONUT, CODI, CCOT, and our CoT2 all aim to reason in continuous space, we propose distinct algorithmic approaches that also address the exploration challenge. Key differences include: (1) Our continuous tokens are simplex-weighted compositions of vocabulary tokens. (2) Our supervision method is novel and explicitly targets implicit parallelism. (3) CoT2 does not initialize from, nor attempt to mimic, discrete CoT. (4) By introducing sampling strategies and associated GRPO variations, we realize the "Supervised Training → Reinforcement Learning" paradigm in the context of CoT2.

The proposed CoT2 approach simultaneously tracks all possible trajectories and superposes them within continuous tokens. This approach is similar to that of Xiong et al. (2024), who superpose multiple candidate outputs into a single final token. Our approach also shares similarities with decoding algorithms like self-consistency (Wang et al., 2022) and Best-of-N-Sampling (Stiennon et al., 2022), which generate multiple trajectories by running inference multiple times and then select a final answer based on the aggregate statistics. In contrast, our algorithm performs a single inference, superposing different trajectories all at once and determining the final answer in one pass. Furthermore, our Dirichlet sampling approach for generating multiple rollouts in GRPO training draws connections to previous works such as Latent Dirichlet Allocation (LDA) (Blei et al., 2003), which introduces Dirichlet priors within a hierarchical Bayesian framework, and AlphaGo (Silver et al., 2017), which injects Dirichlet noise to encourage exploration.

Our work also tangentially relates to research on multi-token prediction (Bachmann & Nagarajan, 2024; Liu et al., 2024; Gloeckle et al., 2024), which aims to improve the efficiency and quality of generation by predicting multiple tokens at once. It is hypothesized that effective future prediction necessitates the exploration of many possible continuations, which is similar to our CoT2 approach.

B. Task Details

B.0.1. Minimum Non-Negative Sum Task

We now introduce the *Minimum Non-Negative Sum* (MNNS) task, where the goal is to assign signs to a list of numbers so that their sum is as small as possible while being nonnegative. The MNNS task can be viewed as partitioning a set of numbers into two subsets with a minimal difference, which makes it closely related to the subset sum problems explored in Dziri et al. (2023); Thomm et al. (2024). Formally, we are given m integers $d_1, \ldots, d_m$, and the task is to assign signs $\sigma_i \in \{+1, -1\}$ such that $s = \sigma_1 d_1 + \cdots + \sigma_m d_m \geq 0$ and s is minimized. Let $\sigma^{\text{opt}} = (\sigma_1^{\text{opt}}, \ldots, \sigma_m^{\text{opt}})$ denote the optimal assignment that achieves the minimal nonnegative sum s^{opt} out of 2^m possible sign assignments. Here, every possible *partial sum* $\sigma_1 d_1 + \cdots + \sigma_t d_t \in \Gamma_t$ is assigned a unique embedding $\boldsymbol{e}_{\phi(\sigma_1 d_1 + \cdots + \sigma_t d_t)}$, where $\phi(\cdot)$ maps each sum to a distinct id in $[v]$. We now describe the two modes of supervision, where the input digits are processed one by one by accumulating partial sums, as illustrated in Figure 1:

- *Supervision for CoT2 model:* At step t, there are $|\Gamma_t| = 2^t$ partial sums of length t, and accordingly, we provide the following target distribution $\boldsymbol{\alpha}_t^*$:

$$\alpha_{t,i}^* = \begin{cases} \text{count}_t(i)/2^t, & \text{if token } i \text{ appears count}_t(i) \text{ times as a partial sum of length } t, \\ 0, & \text{otherwise.} \end{cases}$$

  At the final step $t = m$, the distribution $\boldsymbol{\alpha}_m^*$ assigns probability 1 to the correct sum $\boldsymbol{e}_{\phi(\sigma_1^{\text{opt}} d_1 + \cdots + \sigma_m^{\text{opt}} d_m)}$ and 0 to all others.

- *Supervision for discrete model:* We supervise the discrete model along the correct chain of partial sums by providing $\boldsymbol{e}_{\phi(\sigma_1^{\text{opt}} d_1 + \cdots + \sigma_t^{\text{opt}} d_t)}$ for $1 \leq t \leq m$ as target tokens, and train following the standard cross-entropy objective described in (2).

While constructing the dataset, we split the training and validation sets by ensuring that any permutation of numbers appears in exactly one split. The aim behind this is to prevent memorization and make a fair evaluation. We also encode input and output numbers with separate tokens in our vocabulary. As an example, an input appears as $\langle\text{BOS}\rangle\ d_1\ d_2\ \ldots\ \rightarrow$, and the corresponding output as $s_1\ s_2\ \ldots\ s^{\text{opt}}\ \langle\text{EOS}\rangle$, where s^{opt} is the minimal nonnegative sum for $\{d_1, \ldots, d_m\}$. For the model, we use the GPT2 architecture (Radford et al., 2019) with different head, layer, and embedding dimension configurations, and train it from scratch. Please refer to Appendix G for more experimental details.

B.0.2. ProntoQA and ProsQA Datasets

Other datasets we explore in our investigation of the CSFT approach are the ProntoQA (Saparov & He, 2022) and ProsQA (Hao et al., 2024) datasets, which are logical reasoning tasks that require exploration over multiple possible paths. Specifically, each question in ProntoQA asks whether a certain target word (node) B is reachable from a root word (node) A within a fixed number of hops, while for ProsQA it asks which of the target words B or C is reachable. We use 5-hop questions and present the graph in a structured format. In particular, for each problem, we represent nodes and edges using embeddings, which we use as the model input rather than text input. The detailed structured format and examples are provided in Appendix G.2.

The graph structure of the ProntoQA and ProsQA datasets naturally obeys the supervision in (3), so that we determine the words that can be reached using t edges from A and supervise intermediate tokens on the resulting distribution. At the final reasoning step m, the supervision assigns probability 1 to the correct label: `yes` or `no` for ProntoQA, and `B` or `C` for ProsQA. For the standard discrete model, we provide an explicit chain of nodes from A to the target node (B or C) as the target path at each step. We direct readers to Appendix G.2 for additional details on the supervision.

C. Reinforcement Learning Methods for CoT2

In this section, we describe how to apply RL with continuous output tokens. Specifically, we explore GRPO training on top of continuous or discrete models that are supervised trained based on Section 3 for the MNNS, ProntoQA, and ProsQA tasks. By illustrating two sampling methods for GRPO, we demonstrate how a model trained with discrete SFT can be adapted to produce continuous outputs. We assume a sparse reward setting where the reward is 1 for a correct final answer and 0 otherwise.

Algorithm 1 Multi-Token Sampling GRPO for Continuous Token Generation

Input: Initial policy $\text{LM}_{\theta_{\text{init}}}$; hyperparameters K, G, m, ϵ, β.

$\text{LM}_\theta, \text{LM}_{\theta_{\text{ref}}} \leftarrow \text{LM}_{\theta_{\text{init}}}$
for iteration = 1, 2, . . . , I and **for** step = 1, 2, . . . , S **do**
  Sample a batch of inputs $\{\boldsymbol{X}^{(b)}\}_{b=1}^{B}$
  Update $\text{LM}_{\theta_{\text{old}}} \leftarrow \text{LM}_\theta$
  for each input $\boldsymbol{X}$ in the batch and **for** each trajectory $i = 1, \ldots, G$ from that $\boldsymbol{X}$ **do**
    for each token step $t = 1, \ldots, m$ **do**
      if $t < m$ **then**
        Sample K tokens $\{\boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_K}\}$ from $\boldsymbol{\alpha}_t^{(i),\text{old}}$ to create continuous token $\boldsymbol{z}_t \leftarrow \frac{1}{K}\sum_{r=1}^{K} \boldsymbol{e}_{i_r}$.
        Policy ratio for continuous token $r_t(\theta) \leftarrow \left(\prod_{r=1}^{K} \alpha_{t,i_r}^{(i)} / \prod_{r=1}^{K} \alpha_{t,i_r}^{(i),\text{old}}\right)^{\frac{1}{K}}$.
      else
        Sample $\boldsymbol{z}_m = \boldsymbol{e}_j$ from $\boldsymbol{\alpha}_m^{(i),\text{old}}$.
        Policy ratio for discrete token $r_m(\theta) \leftarrow \alpha_{m,j}^{(i)} / \alpha_{m,j}^{(i),\text{old}}$.
      end if
    end for
    Obtain advantage estimates $\hat{A}_{i,t}$ for each token t in each trajectory $\boldsymbol{Z}^{(i)}$ and calculate objective.
  end for
  Update θ to minimize $\mathcal{L}_{\text{GRPO}}(\theta)$.
end for

Output LM_θ

In our setup, a language model LM_θ acts as a *policy* over tokens. Let $\{\boldsymbol{Z}^{(i)}\}_{i=1}^{G}$ be a group of G trajectories sampled from old policy $\text{LM}_{\theta_{\text{old}}}$ such that each trajectory $\boldsymbol{Z}^{(i)} = \left(\boldsymbol{z}_1^{(i)}, \ldots, \boldsymbol{z}_m^{(i)}\right)$ contains m output tokens given a fixed input $\boldsymbol{X}$. We denote by $\hat{A}_{i,t}$ the advantage estimate at step t in trajectory i and note that $\hat{A}_{i,t} = \hat{A}_i$ is identical across all steps of a trajectory under sparse reward setting. To quantify how the new policy LM_θ differs from the old one on token $\boldsymbol{z}_t^{(i)}$ from ith trajectory, we define the policy ratio $r_t^{(i)}(\theta) = \frac{\text{LM}_\theta\left(\boldsymbol{z}_t^{(i)} | \boldsymbol{z}_{<t}^{(i)}, \boldsymbol{X}\right)}{\text{LM}_{\theta_{\text{old}}}\left(\boldsymbol{z}_t^{(i)} | \boldsymbol{z}_{<t}^{(i)}, \boldsymbol{X}\right)}$. We update the model parameters θ by minimizing the clipped surrogate objective (Shao et al., 2024; Yu et al., 2025):

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{\sum_{i=1}^{G} |\boldsymbol{Z}^{(i)}|} \sum_{i=1}^{G} \sum_{t=1}^{|\boldsymbol{Z}^{(i)}|} \left[\min\left(r_t^{(i)}(\theta)\,\hat{A}_{i,t},\ \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\,\hat{A}_{i,t}\right) - \beta\,\mathbb{D}_{\text{KL}}\left[\text{LM}_\theta \,\|\, \text{LM}_{\theta_{\text{ref}}}\right]\right].$$

As the output length is fixed in our setting, we have $|\boldsymbol{Z}^{(i)}| = m$ for each trajectory. Here, ϵ is a clipping parameter that bounds the ratio $r_t(\theta)$, and β controls the strength of KL-divergence from a reference policy $\text{LM}_{\theta_{\text{ref}}}$ which is the SFT-initialized policy. We set the number of GRPO iterations $\mu = 1$ and estimate the KL divergence with the Schulman Approximator as in Shao et al. (2024).

C.1. Multi-Token Sampling

We emulate the rollout of a continuous token by sampling a fixed number of $K \leq v$ discrete tokens and averaging them at steps $t = 1, \ldots, m-1$. We refer to this hybrid method as *CoT2 with MTS* (multi-token sampling). For the GRPO objective, we propose the following method to calculate the policy ratio for continuous tokens. Specifically, assume at step t we sample discrete tokens $\boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_K}$ with probabilities $\alpha_{t,i_1}, \ldots, \alpha_{t,i_K}$ under the current policy and probabilities $\alpha_{t,i_1}^{\text{old}}, \ldots, \alpha_{t,i_K}^{\text{old}}$ under the old policy. We define the policy ratio for these continuous steps by dividing geometric means:

$$r_t(\theta) = \frac{\text{LM}_\theta\left(\boldsymbol{z}_t \mid \boldsymbol{z}_{<t}, \boldsymbol{X}\right)}{\text{LM}_{\theta_{\text{old}}}\left(\boldsymbol{z}_t \mid \boldsymbol{z}_{<t}, \boldsymbol{X}\right)} = \left(\frac{\alpha_{t,i_1} \cdots \alpha_{t,i_K}}{\alpha_{t,i_1}^{\text{old}} \cdots \alpha_{t,i_K}^{\text{old}}}\right)^{1/K} \quad \text{for } t = 1, \ldots, m-1. \tag{4}$$

The geometric mean ensures that the ratio for each continuous step remains on the same scale as the final discrete token's ratio and, thus, helps avoid overly large or small updates in the GRPO objective and provides more stable training compared to the direct multiplication of probabilities. Once this ratio is computed, we then average the K sampled tokens to form $\boldsymbol{z}_t$, which is fed to the model as the query for the next prediction step. At the final step $t = m$, where the token $\boldsymbol{z}_m = \boldsymbol{e}_j$ is discrete with $j \in [v]$ denoting its index, the policy ratio is simply the probability ratio of selecting that token:

$$r_m(\theta) = \frac{\text{LM}_\theta\left(\boldsymbol{z}_m \mid \boldsymbol{z}_{<m}, \boldsymbol{X}\right)}{\text{LM}_{\theta_{\text{old}}}\left(\boldsymbol{z}_m \mid \boldsymbol{z}_{<m}, \boldsymbol{X}\right)} = \frac{\alpha_{m,j}}{\alpha_{m,j}^{\text{old}}}. \tag{5}$$

Table 2. Validation accuracy and token-level entropy of CoT2 with MTS on the discrete model under different rollout sizes K for MNNS task. We use 4 input digits in 1-9 with 1-layer, 1-head GPT2 at embedding dimensions 24 and 32, with SFT accuracies of 39.76% and 43.50%, respectively.

	K	Val. Accuracy (%)		Val. Entropy (SFT → SFT+GRPO)			
		SFT	SFT+GRPO	token_1	token_2	token_3	token_4
24	1	39.76	49.01	0.3218 → 0.0314	0.5858 → 0.0712	0.5499 → 0.1576	0.4786 → 0.1718
	3		52.60	0.3717 → 0.0647	0.7461 → 0.2120	0.8006 → 0.3312	0.5338 → 0.1529
	6		49.69	0.4524 → 0.1242	0.7738 → 0.3361	0.8364 → 0.6615	0.5134 → 0.2159
32	1	43.50	51.61	0.3618 → 0.0143	0.6331 → 0.0395	0.3518 → 0.2631	0.1962 → 0.1182
	3		55.66	0.3886 → 0.0412	0.7101 → 0.0904	0.5447 → 0.5757	0.2863 → 0.1734
	6		50.38	0.4224 → 0.0632	0.7915 → 0.2161	0.6077 → 0.8481	0.2780 → 0.1514

Inference. During inference after GRPO training, we apply the same multi-token sampling procedure at each of the first $m-1$ steps to form the continuous token via the average of K sampled embeddings.

Remark. An alternative to the normalization of ratios given by (4) is to directly scale down the logits by $1/K$ before applying softmax. However, using this approach during inference leads to a distribution shift relative to the SFT-trained model and ultimately degrades performance.

C.2. Dirichlet Sampling

In this section, we present another method for generating continuous tokens at each step by interpreting the model's output distribution $\boldsymbol{\alpha}_t \in \Delta^{v-1}$ as concentration parameters of a Dirichlet distribution over the $v-1$ simplex. We introduce a scaling hyperparameter $\gamma > 0$ and define the Dirichlet distribution with the parameters $\gamma\,\boldsymbol{\alpha}_t = (\gamma\alpha_{t,1}, \ldots, \gamma\alpha_{t,v})$. Without this scaling, directly using $\boldsymbol{\alpha}_t$ as parameters often causes training instability, particularly when many $\alpha_{t,i}$ values are small. We then sample a point $\hat{\boldsymbol{\alpha}}_t \in \Delta^{v-1}$ from the resulting distribution $\mathrm{Dir}(\boldsymbol{\alpha}_t)$. After sampling, we form the continuous token by mapping $z_t = \boldsymbol{E}^\top \hat{\boldsymbol{\alpha}}_t \in \mathbb{R}^d$, which becomes the query for the next step. We denote the Dirichlet densities induced by current and old policies as $f_\theta(z_t)$ and $f_{\theta_{\text{old}}}(z_t)$, respectively. Accordingly, we define the policy ratio at a continuous step $t < m$ as:

$$r_t(\theta) = \frac{\mathrm{LM}_\theta(z_t \mid z_{<t}, \boldsymbol{X})}{\mathrm{LM}_{\theta_{\text{old}}}(z_t \mid z_{<t}, \boldsymbol{X})} = \frac{f_\theta(z_t)}{f_{\theta_{\text{old}}}(z_t)},$$

The above definition parallels how we compute probability ratios for discrete actions but replace the categorical pmf with continuous Dirichlet pdf. At the final step $t = m$, we sample a discrete token $z_m \in \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_v\}$ from $\boldsymbol{\alpha}_m$, and use the standard policy ratio given by (5). At inference, we follow the autoregressive procedure in Section 3 by creating a convex combination of vocabulary tokens.

C.3. Results and Discussion

Table 1 demonstrates that for each $K \in \{1, 3, 6\}$, CoT2 with MTS significantly improves validation accuracy relative to the discrete SFT baseline (39.76%), with moderate K yielding the best final performance. We also observe that smaller K-values correspond to larger reductions in token-level entropies, suggesting that the model becomes more confident at each intermediate step by learning to commit fewer tokens. Notably, this suggests a curriculum on K—starting small and gradually increasing—could potentially further improve the training. Interestingly, the third token's entropy remains relatively high, which might indicate that the model continues to hedge among several partial expansions at that step, which may help preserve useful diversity of reasoning. Therefore, CoT2 with MTS enables a model trained with discrete SFT to produce continuous outputs and helps it achieve better performance. We further discuss Dirichlet sampling in Appendix E.2 and show that the accuracy of the CoT2 model can be further improved-despite already achieving near-perfect performance once the embedding dimension is sufficiently large.

D. Theoretical Analysis

In this section, we first present the construction of a single-layer transformer that solves the MNNS task using an attention layer followed by a mixture-of-experts MLP layer. We then provide a theoretical comparison between base CoT2, CoT2-

MTS, and discrete CoT models.

Proposition 1. *There exists a* 1*-layer transformer architecture with a mixture-of-experts MLP layer that solves the MNNS task using CoT2 and following the CSFT supervision in Section 3.*

The above construction utilizes trigonometric embeddings, inspired by the mechanistic insights given by Nanda et al. (2023). In contrast to their focus, our approach leverages these trigonometric embeddings to give a theoretical guarantee that the transformer can simultaneously track multiple numbers by benefiting from the embedding capacity and read off the minimum non-negative number at the final step. An important observation regarding the construction of the solution is that this construction decouples the different trajectories at each intermediate reasoning step. Specifically, it satisfies the following Assumption 1 as shown in Appendix F. Building on that insight, we adopt this property to compare CoT2 and discrete CoT models in the remainder of this section.

Assumption 1. *Recall the model* LM_θ *in Section 2. For any step* t *and prefix tokens* $z_{<t}$*. We assume (i) the next token probabilities depend only on the last token* z_t *and (ii) if the query token is* $z_t = \sum_{j=1}^{v} \alpha_{t,j}\, \boldsymbol{e}_j$ *so that* $\sum_{j=1}^{v} \alpha_{t,j} = 1$*, the output distribution decouples as follows:*

$$\mathrm{LM}_\theta^{(t)}(\cdot \mid z_t, \boldsymbol{X}) \overset{(i)}{=} \mathrm{LM}_\theta^{(t)}(\cdot \mid z_{\leq t}, \boldsymbol{X}) \overset{(ii)}{=} \sum_{j=1}^{v} \alpha_{t,j}\, \mathrm{LM}_\theta^{(t)}(\cdot \mid \boldsymbol{e}_j, \boldsymbol{X}).$$

Independent of Assumption 1, the token distribution $\boldsymbol{\alpha}_t = \mathrm{LM}_\theta^{(t)}(\cdot \mid z_{<t}, \boldsymbol{X})$ evolves with the equation $\boldsymbol{\alpha}_t = \boldsymbol{\alpha}_{t-1} \boldsymbol{M}_t(z_{<t}; \boldsymbol{X})$, starting from $\boldsymbol{\alpha}_0 = \mathrm{LM}_\theta^{(0)}(\cdot \mid \boldsymbol{X})$ until $\boldsymbol{\alpha}_m$. Here, $\boldsymbol{M}_t(z_{<t}; \boldsymbol{X}) \in \mathbb{R}^{v \times v}$ is a context-conditioned transition matrix that depends on the inputs $\boldsymbol{X}$ and the previous tokens $z_{<t}$ (Ildiz et al., 2024). For the simplicity of notation, we omit $z_{<t}$ and $\boldsymbol{X}$ in the notation of $\boldsymbol{M}_t(z_{<t}; \boldsymbol{X})$, and use $\boldsymbol{M}_t$.

- ***Base CoT2 model***: At each step $t = 1, \ldots, m$, the model outputs the continuous token $z_t = \boldsymbol{E}^\top \boldsymbol{\alpha}_{t-1}$ and uses it as the query for the next step.
- ***Discrete CoT model***: At each step $1 \leq t \leq m$, the model samples exactly one token $z_t = \boldsymbol{e}_{i_t}$ from $\boldsymbol{\alpha}_t$, and uses it as the query for the next step.
- ***CoT2 with MTS (multi-token sampling)***: At each step $1 \leq t \leq m$, i.i.d. sample K tokens $\boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_K}$ from to $\boldsymbol{\alpha}_t$, average these tokens to form $z_t = \frac{1}{K} \sum_{r=1}^{K} \boldsymbol{e}_{i_r}$, which it uses as query for the next step.

With these three approaches defined, we present the following result on the consistency of the models.

Proposition 2 (Consistency of Models)**.** *Under Assumption 1 above, given any input* $\boldsymbol{X}$*, the conditional distributions at the final step* $\boldsymbol{\alpha}_m = \boldsymbol{\alpha}_0 \prod_{t=1}^{m} \boldsymbol{M}_t$ *obtained from the above three models yield the same final token distribution.*

Proposition 2 indicates that discrete, continuous, and multi-token sampling CoT models are consistent in the final distribution. However, they differ in how many samples are needed to approximate that distribution with repeated samplings. In the following lemma, we explore the sample complexity needed to recover the output distribution based on the ℓ_1, or equivalently, TV (Total-Variation) error.

Proposition 3. *Let* $\boldsymbol{\alpha}_m$ *be the final output distribution after* m *steps of CoT. Let repeated CoT2 with MTS samplings form the empirical distribution* $\hat{\boldsymbol{\alpha}}_m$ *over the output distribution. Then, to ensure* $\|\hat{\boldsymbol{\alpha}}_m - \boldsymbol{\alpha}_m\|_1 \leq \epsilon$ *with high probability, the total number of samples required scales as* $\Theta\left(\frac{v}{K\,\epsilon^2}\right)$.

The above proposition characterizes the sample complexity of CoT2 with MTS, which generalizes discrete CoT ($K = 1$). For $K = 1$, which corresponds to the discrete CoT model, the above proposition reduces to the known $\Theta(\frac{v}{\epsilon^2})$ sample complexity of approximating a v-category distribution in ℓ^1 distance (Kamath et al., 2015). Thus, although the three models yield the same final distribution, the discrete model requires substantially more rollouts for accurate approximation due to inherent noise from single-token sampling. In contrast, the CoT2 model carries the entire mixture of partial expansions at each step and computes the distribution in one shot, while CoT2 with MTS model captures multiple partial expansions in each step and proportionally reduces the sample complexity.

E. Experimental Results

E.1. Continuous Supervised Training Results

Teacher Forcing and Self-feeding Comparison: As described in Section 3, we tested two approaches of providing prefixes during training the CoT2 model with CSFT. Although the model autoregressively generates at inference time, teacher forcing yields better performance than self-feeding during CSFT training. Our results demonstrate that, We also tested curriculum settings, where we switch to self-feeding after a pre-determined number of epochs in the training. Still, the accuracies didn't improve beyond pure teacher-forcing training. The results are illustrated in Figure 4, where we refer to teacher-forcing as "hard-teacher" and refer to self-feeding as "soft-teacher".

Sparse Supervision for Discrete Baseline: We also tested providing a subset of the correct path to the discrete model. We observed that a sparsely supervised discrete model can achieve better performance than the fully supervised discrete model when the distribution is "easier" to handle by the model. As an example, we tested the case when we have 5 input digits from the range of 11 to 19. In this case, in nearly all of the cases, the answer to our MNNS game is (sum of minimum 3 numbers) - (sum of maximum 2 numbers) out of the 5 input numbers. In this case, when only 1 token from the correct path is provided to the discrete model, it's better than 3 and 5 token cases. However, when we change the distribution to a range of numbers from 5 to 13, which makes the question reasonably harder, the discrete model with 1 token supervision performs worse than the other two, and the discrete model with full supervision performs best. The results are demonstrated in Figure 3.

Further Results on CoT2 vs Discrete CoT: The results in Figure 5 also indicate that above an embedding dimension threshold, the CoT2 model has superior performance and trains significantly faster than the discrete CoT model. Moreover, combining the results of Figure 5 with Figure 2c, we see that the CoT2 model with one layer and one head GPT2 model performs better than discrete CoT model with two layers and two heads at embeddings 24 and 32. While the continuous approach requires greater embedding capacity to support its distributional representations at each step, it can outperform the discrete model using fewer layers and attention heads. Supporting the findings in Figure 2, Figure 6 illustrates that on the ProntoQA task, CoT2 consistently outperforms the discrete CoT baseline when the embedding dimension is above a threshold. Likewise, as depicted in Figure 7 and Figure 8, the discrete CoT model requires multiple samplings (Maj@k) to match the single-shot performance of CoT2 on both ProntoQA and ProsQA, which indicates that CoT2 model is more sample-efficient.

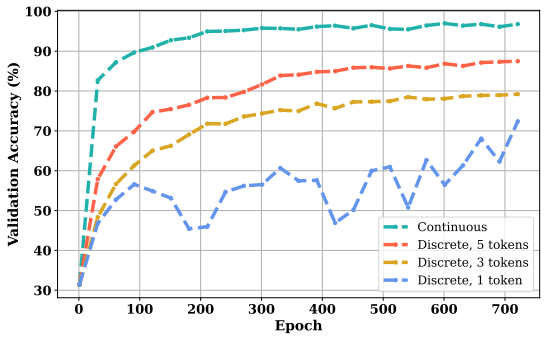


Figure 3. The figure illustrates that when the range of digits makes the question non-trivial on an MNNS task, the discrete CoT model trained with full token supervision outperforms sparse supervisions; in particular, single token supervision yields the worst performance. **Setting:** 5 input digits in $5-13$; 2-layer, 2-head GPT2 with $d = 32$.

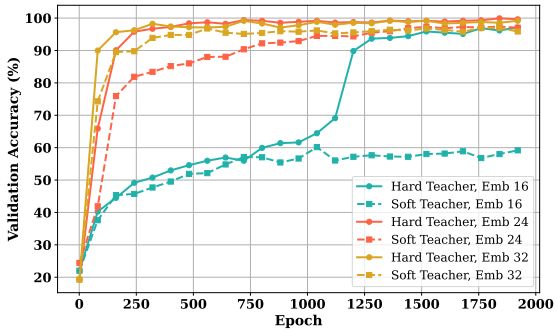


Figure 4. The comparison between the hard and soft teachers for different embedding dimensions. The figure illustrates that the hard teacher is superior to the soft teacher. **Setting:** 4 input digits in $1-9$; 4-layer, 4-head GPT2 with $d \in \{16, 24, 32\}$.

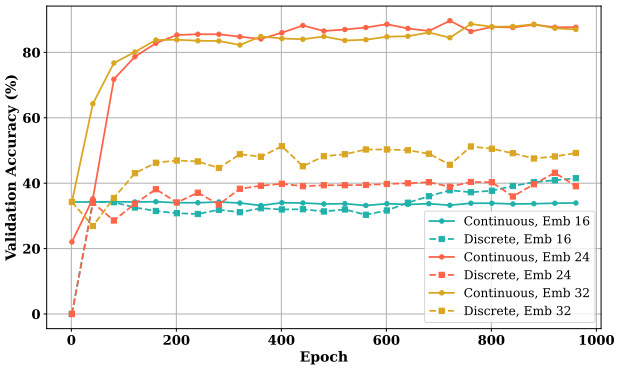


Figure 5. Comparison between CoT2 and discrete CoT2 model for different embedding dimensions. The figure demonstrates that above a certain embedding dimension threshold, the CoT2 model outperforms the discrete CoT model in the MNNS task. **Setting:** 4 input digits in $1-9$; 1-layer, 1-head GPT2 with $d \in \{16, 24, 32\}$.

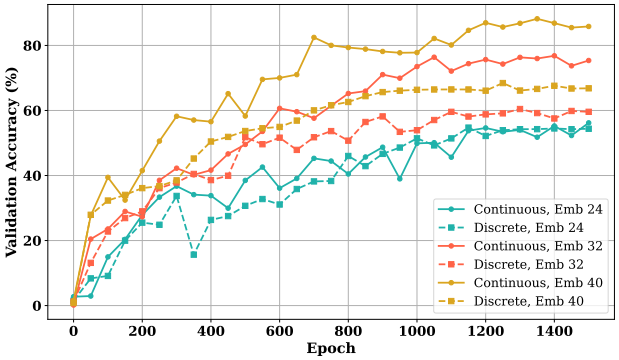


Figure 6. Comparison between CoT2 and discrete CoT2 model for different embedding dimensions in ProntoQA task. The figure shows that above an embedding dimension threshold, the CoT2 model outperforms the discrete CoT model. **Setting:** 4 input digits in $1-9$; 4-layer, 4-head GPT2 with $d \in \{24, 32, 30\}$.

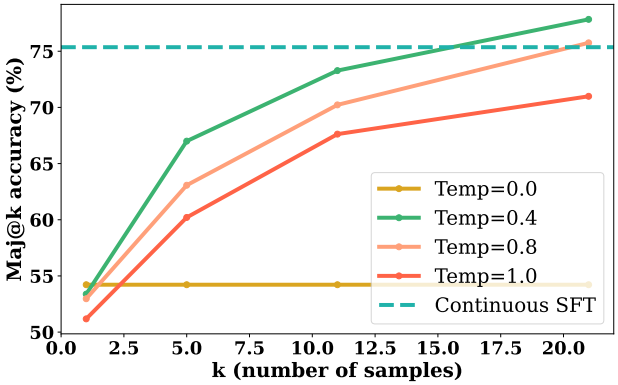


Figure 7. The figure illustrates that the discrete CoT2 model requires multiple samplings (Maj@k) to match the single-shot performance of the CoT2 model on ProntoQA (10-run average). **Setting:** 4-layer, 4-head GPT2 with $d = 32$.

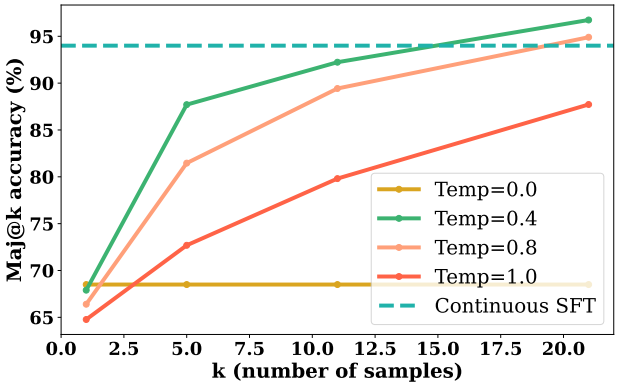


Figure 8. The figure illustrates that the discrete CoT2 model requires multiple samplings (Maj@k) to match the single-shot performance of the CoT2 model on ProsQA (10-run average). **Setting:** 4-layer, 4-head GPT2 with $d = 32$.

E.2. GRPO Results

Discussion on ProntoQA/ProsQA Datasets: Table 3 illustrates that GRPO training using CoT2 with MTS sampling consistently improves discrete CoT and CoT2 models over their initial SFT accuracy. Moreover, we observe that the improvement in the discrete CoT model is greater, which might indicate that the CoT2 model already gains an RL-like exploration mechanism through CSFT training. We observe that while increasing K initially increases the accuracy by sampling more tokens at each step, beyond some point it degrades performance. This observation is consistent with Table 1, where we see that moderate K values offer the best final performance. One possible explanation is that while higher K promotes better exploration, it also raises the chance of sampling unhelpful tokens that disrupt the averaged token representation. Indeed, for larger K, we observe that the RL objective saturates to near zero which suggests that most rollouts fail once the averaged token contains too many distracting tokens.

Discussion on Dirichlet Sampling: We also investigate the effects of Dirichlet sampling in GRPO training discrete CoT and CoT2 models. The results in Table 4 indicates that applying Dirichlet sampling ($\gamma = 20$) in GRPO training of discrete CoT model consistently improves over the initial SFT training accuracies. Similar to the CoT2 + MTS sampling results in Table 1, we observe that the entropy at the third token remains relatively high, which suggests a beneficial diversity in model's predictions for that token. Moreover, the Table 5 demonstrates that Dirichlet sampling also improves the CoT2 model's SFT accuracy, even though it has a high initial SFT accuracy. As illustrated in Table 5, we find there is an optimal value for the scale parameter γ, since larger γ typically yields more uniform sampling distributions, whereas smaller γ concentrates the distribution more sharply. Thus, adjusting γ provides a balance between exploration and stability in GRPO training.

Table 3. Validation accuracies on ProsQA and ProntoQA for CoT2 and Discrete CoT, evaluated at $K = 6, 8, 12$ using CoT2 with MTS sampling GRPO. All models use a 4-layer, 4-head GPT2 with embedding dimension 32. The SFT values are constant as they represent initial accuracies before GRPO.

		ProsQA		ProntoQA	
		SFT	SFT+GRPO	SFT	SFT+GRPO
K=6	CoT2	93.37	93.83	75.36	76.15
	Discrete CoT	68.50	68.24	59.58	62.28
K=8	CoT2	93.37	94.09	75.36	76.66
	Discrete CoT	68.50	71.58	59.58	71.53
K=12	CoT2	93.37	94.21	75.36	77.64
	Discrete CoT	68.50	72.76	59.58	74.03

Table 4. Discrete CoT models trained with GRPO after SFT using Dirichlet sampling ($\gamma = 20$) and a learning rate of 1×10^{-5}. We show validation accuracy (%) and token-level entropy (SFT → SFT+GRPO) for each (Layers, Heads) setting, with an embedding dimension of 24 for GPT2 model.

Layers	Heads	Val. Accuracy (%)		Val. Entropy (SFT → SFT+GRPO)			
		SFT	SFT+GRPO	token_1	token_2	token_3	token_4
1	1	39.76	46.25	0.4851 → 0.1701	0.5165 → 0.6380	0.3243 → 0.6590	0.1597 → 0.4878
2	2	70.26	75.84	0.4851 → 0.4027	0.5165 → 0.4413	0.3243 → 0.2907	0.1597 → 0.1386

Table 5. Validation accuracies GRPO training with CoT2 models using different Dirichlet sampling scales (γ) with learning rate of 1×10^{-6}. We show the baseline SFT accuracy (87.84%) and final performance after GRPO.

Dirichlet Scale (γ)	SFT Val. Acc (%)	SFT + GRPO Val. Acc (%)
10		89.76
20	87.84	90.75
40		90.37

F. Further Theoretical Details

Justification for Assumption 1: The Assumption 1 holds for tasks where the next-token distribution depends solely on the current token and the input tokens rather than the full history of output tokens. This is satisfied by many reasoning tasks, where the aim is to keep track of an intermediate state (e.g., the current sum) and update this state based only on the current state and the input, independently of the earlier trajectory.

For example, in the MNNS task, the model generates a token representing the current partial sum at each step. To compute the distribution over the next possible sums, the model adds or subtracts the selected number from the input context $\boldsymbol{X}$ to the current sum, without needing to remember the sequence of previous sums explicitly. Thus, the next-state distribution at each step is only determined by the current state and it naturally satisfies the Assumption 1.

Proposition 2 (Consistency of Models)**.** *Under Assumption 1 above, given any input $\boldsymbol{X}$, the conditional distributions at the final step $\boldsymbol{\alpha}_m = \boldsymbol{\alpha}_0 \prod_{t=1}^{m} \boldsymbol{M}_t$ obtained from the above three models yield the same final token distribution.*

Remark: Please note that $\boldsymbol{\alpha}_m$ represents a distribution of tokens. This proposition argues that the distributions of $\boldsymbol{\alpha}_m$ obtained from the three models converges to the same model as the number of sampling approaches infinity. Specifically, as the number of samples approaches infinity, the distributions of $\boldsymbol{\alpha}_m$ obtained from the discrete CoT and CoT2 with MTS models converges in probability to $\boldsymbol{\alpha}_m$ obtained from the CoT2 model. An important remark is that $\boldsymbol{\alpha}_m$ obtained from CoT2 model is not a random variable as this parameter obtained in a sample-free way.

Proof. Let $\hat{\boldsymbol{\alpha}}_t^{(\text{disc})}, \hat{\boldsymbol{\alpha}}_t^{(\text{MTS})}$ denote the empirical output token distributions at step t under one trajectory obtained by the discrete CoT, and CoT2 with MTS models, respectively. We define $\boldsymbol{\alpha}_t^{(\text{disc})} = \mathbb{E}\left[\hat{\boldsymbol{\alpha}}_t^{(\text{disc})}\right]$ and $\boldsymbol{\alpha}_t^{(\text{MTS})} = \mathbb{E}\left[\hat{\boldsymbol{\alpha}}_t^{(\text{MTS})}\right]$ to be corresponding expected distributions. The discrete CoT model at each step picks exactly 1 token from $\boldsymbol{\alpha}_t$. On the other hand, CoT2 with MTS samples K i.i.d. tokens at every step independently according to their probabilities from $\hat{\boldsymbol{\alpha}}_t$. We denote them $i_1, \ldots, i_K$, and average their embeddings to produce a single query.

We will use induction in our argument. For the base case, all models start with the same initial distribution, so we trivially have $\boldsymbol{\alpha}_0^{(\text{disc})} = \boldsymbol{\alpha}_0^{(\text{MTS})} = \boldsymbol{\alpha}_0^{(\text{CoT2})}$. For the inductive step, assume that we have $\boldsymbol{\alpha}_{t-1}^{(\text{disc})} = \boldsymbol{\alpha}_{t-1}^{(\text{MTS})} = \boldsymbol{\alpha}_{t-1}^{(\text{CoT2})}$. We will show that $\boldsymbol{\alpha}_t^{(\text{disc})} = \boldsymbol{\alpha}_t^{(\text{MTS})} = \boldsymbol{\alpha}_t^{(\text{CoT2})}$. On the other, for the discrete CoT model, the model samples one token $\boldsymbol{e}_{i_{t+1}}$ from the row of $\boldsymbol{M}_t$ for a token i_{t+1}. Therefore, we need to condition on the token at step t. We have:

$$
\begin{aligned}
\mathbb{E}\left[\hat{\boldsymbol{\alpha}}_{t+1}^{(\text{disc})}\right] = \sum_{j=1}^{v} \mathbb{P}(z_t = \boldsymbol{e}_j)\, \mathbb{E}\left[\hat{\boldsymbol{\alpha}}_{t+1}^{(\text{disc})} \mid z_t = \boldsymbol{e}_j\right] &= \sum_{j=1}^{v} \mathbb{P}(z_t = \boldsymbol{e}_j)\, \mathrm{LM}_\theta^{(t)}(\cdot \mid \boldsymbol{e}_j, \boldsymbol{X}) \\
&\sum_{j=1}^{v} \alpha_{t,j}^{(\text{disc})}\, \mathrm{LM}_\theta^{(t)}(\cdot \mid \boldsymbol{e}_j, \boldsymbol{X}) \\
&\overset{(a)}{=} \sum_{j=1}^{v} \alpha_{t,j}^{(\text{CoT2})}\, \mathrm{LM}_\theta^{(t)}(\cdot \mid \boldsymbol{e}_j, \boldsymbol{X}) \\
&\overset{(b)}{=} \boldsymbol{\alpha}_{t+1}^{(\text{CoT2})}
\end{aligned}
\tag{6}
$$

where (a) follows from the induction argument and (b) follows from Assumption 1. Therefore, we obtain $\boldsymbol{\alpha}_{t+1}^{(\text{disc})} = \mathbb{E}\left[\hat{\boldsymbol{\alpha}}_{t+1}^{(\text{disc})}\right] = \boldsymbol{\alpha}_{t+1}^{(\text{CoT2})}$. For the CoT2 with MTS model, the argument will be similar. Using the decoupling of trajectories by Assumption 1, the next distribution is:

$$
\mathrm{LM}_\theta^{(t)}\left(\cdot \mid \frac{1}{K}\sum_{r=1}^{K} \boldsymbol{e}_{i_r}, \boldsymbol{X}\right) = \frac{1}{K}\sum_{r=1}^{K} \mathrm{LM}_\theta^{(t)}\left(\cdot \mid \boldsymbol{e}_{i_r}, \boldsymbol{X}\right).
$$

Therefore, we write:

$$
\begin{aligned}
\mathbb{E}\left[\hat{\boldsymbol{\alpha}}_{t+1}^{(\mathrm{MTS})}\right] &= \sum_{(i_1,\ldots,i_K)\in[v]^K} \mathbb{P}(\boldsymbol{e}_{i_1},\ldots,\boldsymbol{e}_{i_K})\,\mathbb{E}\left[\hat{\boldsymbol{\alpha}}_{t+1}^{(\mathrm{MTS})} \mid \boldsymbol{e}_{i_1},\ldots,\boldsymbol{e}_{i_K}\right] \\
&= \sum_{(i_1,\ldots,i_K)\in[v]^K} \mathbb{P}(\boldsymbol{e}_{i_1},\ldots,\boldsymbol{e}_{i_K})\,\mathrm{LM}_\theta^{(t)}\left(\cdot \mid \frac{1}{K}\sum_{r=1}^{K}\boldsymbol{e}_{i_r}, \boldsymbol{X}\right) \\
&= \sum_{(i_1,\ldots,i_K)\in[v]^K} \mathbb{P}(\boldsymbol{e}_{i_1},\ldots,\boldsymbol{e}_{i_K})\frac{1}{K}\sum_{r=1}^{K}\mathrm{LM}_\theta^{(t)}\left(\cdot \mid \boldsymbol{e}_{i_r}, \boldsymbol{X}\right) \\
&= \sum_{(i_1,\ldots,i_K)\in[v]^K} \left(\prod_{r=1}^{K}\mathbb{P}(\boldsymbol{e}_{i_r})\right)\frac{1}{K}\sum_{r=1}^{K}\mathrm{LM}_\theta^{(t)}\left(\cdot \mid \boldsymbol{e}_{i_r}, \boldsymbol{X}\right) \\
&= \sum_{(i_1,\ldots,i_K)\in[v]^K} \left(\prod_{r=1}^{K}\alpha_{t,i_r}^{(\mathrm{MTS})}\right)\frac{1}{K}\sum_{r=1}^{K}\mathrm{LM}_\theta^{(t)}\left(\cdot \mid \boldsymbol{e}_{i_r}, \boldsymbol{X}\right) \\
&= \frac{1}{K}\sum_{r=1}^{K}\sum_{j=1}^{v}\mathrm{LM}_\theta^{(t)}(\cdot \mid \boldsymbol{e}_j, \boldsymbol{X}) \sum_{(i_1,\ldots,i_{r-1},i_{r+1},\ldots,i_K)\in[v]^{K-1}} \alpha_{t,j}^{(\mathrm{MTS})}\prod_{\substack{s=1\\ s\neq r}}^{K}\alpha_{t,i_s}^{(\mathrm{MTS})} \\
&= \frac{1}{K}\sum_{r=1}^{K}\sum_{j=1}^{v}\alpha_{t,j}^{(\mathrm{MTS})}\,\mathrm{LM}_\theta^{(t)}(\cdot \mid \boldsymbol{e}_j, \boldsymbol{X}) \sum_{(i_1,\ldots,i_{r-1},i_{r+1},\ldots,i_K)\in[v]^{K-1}} \prod_{\substack{s=1\\ s\neq r}}^{K}\alpha_{t,i_s}^{(\mathrm{MTS})} \\
&= \sum_{r=1}^{K}\frac{1}{K}\sum_{j=1}^{v}\alpha_{t,j}^{(\mathrm{MTS})}\,\mathrm{LM}_\theta^{(t)}\left(\cdot \mid \boldsymbol{e}_j, \boldsymbol{X}\right) = \sum_{j=1}^{v}\alpha_{t,j}^{(\mathrm{MTS})}\,\mathrm{LM}_\theta^{(t)}\left(\cdot \mid \boldsymbol{e}_j, \boldsymbol{X}\right) \\
&= \sum_{j=1}^{v}\alpha_{t,j}^{(\mathrm{CoT2})}\,\mathrm{LM}_\theta^{(t)}\left(\cdot \mid \boldsymbol{e}_j, \boldsymbol{X}\right) = \boldsymbol{\alpha}_{t+1}^{(\mathrm{CoT2})}. \qquad (7)
\end{aligned}
$$

Therefore, combining (6), and (7) completes our argument:

$$
\boldsymbol{\alpha}_{t+1}^{(\mathrm{disc})} = \mathbb{E}\left[\hat{\boldsymbol{\alpha}}_{t+1}^{(\mathrm{disc})}\right] = \boldsymbol{\alpha}_{t+1}^{(\mathrm{CoT2})} = \mathbb{E}\left[\hat{\boldsymbol{\alpha}}_{t+1}^{(\mathrm{MTS})}\right] = \boldsymbol{\alpha}_{t+1}^{(\mathrm{MTS})}.
$$

□

Proposition 3. *Let $\boldsymbol{\alpha}_m$ be the final output distribution after m steps of CoT. Let repeated CoT2 with MTS samplings form the empirical distribution $\hat{\boldsymbol{\alpha}}_m$ over the output distribution. Then, to ensure $\|\hat{\boldsymbol{\alpha}}_m - \boldsymbol{\alpha}_m\|_1 \leq \epsilon$ with high probability, the total number of samples required scales as $\Theta\left(\frac{v}{K\epsilon^2}\right)$.*

Proof. We will utilize the empirical distributions $\hat{\boldsymbol{\alpha}}_t^{(\mathrm{disc})}, \hat{\boldsymbol{\alpha}}_t^{(\mathrm{MTS})}$ that are defined in the proof of Proposition 3 and show that i.i.d. sampling K discrete CoT trajectories and averaging their results at the last step is distributionally equivalent to CoT2 with MTS using K tokens, under Assumption 1. We will first argue the results for $m = 1$ and then we will show it for any m. As discussed in the previous proposition, using the decoupling of trajectories by Assumption 1, the next distribution $\hat{\boldsymbol{\alpha}}_1^{(\mathrm{MTS})}$ when $\boldsymbol{e}_{i_1},\ldots,\boldsymbol{e}_{i_K}$ are drawn from $\boldsymbol{\alpha}_0$ is:

$$
\begin{aligned}
\hat{\boldsymbol{\alpha}}_1^{(\mathrm{MTS})} &= \mathrm{LM}_\theta^{(1)}\left(\cdot \mid \frac{1}{K}\sum_{r=1}^{K}\boldsymbol{e}_{i_r}, \boldsymbol{X}\right) \\
&= \frac{1}{K}\sum_{r=1}^{K}\mathrm{LM}_\theta^{(1)}\left(\cdot \mid \boldsymbol{e}_{i_r}, \boldsymbol{X}\right) \\
&= \frac{\sum_{r=1}^{K}\mathrm{LM}_\theta^{(1)}\left(\cdot \mid \boldsymbol{e}_{i_r}, \boldsymbol{X}\right)}{K} \\
&= \frac{\sum_{r=1}^{K}\hat{\boldsymbol{\alpha}}_{1,r}^{(\mathrm{disc})}}{K}, \qquad (8)
\end{aligned}
$$

which is the empirical mean of K i.i.d. discrete CoT draws. Thus, under our assumption, drawing K tokens with the MTS approach is distributionally equivalent to combining outcomes of K independent discrete CoT draws. Now, for any t, we know by Assumption 1 that output at step $t+1$ does not depend on the previous history of tokens. Because of this, we only focus on the tokens $\boldsymbol{e}_{i_1}, \ldots, \boldsymbol{e}_{i_K}$ drawn at step t from $\boldsymbol{\alpha}_t^{(\mathrm{MTS})}$, where $\boldsymbol{\alpha}_t^{(\mathrm{MTS})} = \boldsymbol{\alpha}_t^{(\mathrm{disc})}$ according to Proposition 2. Following the same steps as in (8), we obtain that:

$$\hat{\boldsymbol{\alpha}}_{t+1}^{(\mathrm{MTS})} = \frac{\sum_{r=1}^{K} \mathrm{LM}_\theta^{(t)}(\cdot \mid \boldsymbol{e}_{i_r}, \boldsymbol{X})}{K} = \frac{\sum_{r=1}^{K} \hat{\boldsymbol{\alpha}}_{t+1,r}^{(\mathrm{disc})}}{K},$$

which is again the empirical mean of K i.i.d. discrete CoT trajectory output token distributions. To finish our argument, we will benefit from a standard result in multinomial estimation that $\Theta\left(\frac{v}{\epsilon^2}\right)$ i.i.d. samples are necessary and sufficient to learn a v-category distribution in $\|\cdot\|_1$-distance $\leq \epsilon$ (Kamath et al., 2015). In our MTS setting, each sample uses K i.i.d. draws internally. This reduces the variance by a factor of K and achieves the same estimation guarantee with $\Theta\left(\frac{v}{K\epsilon^2}\right)$ aggregated samples. Hence the total sample complexity in terms of MTS samplings is $\Theta\left(\frac{v}{K\epsilon^2}\right)$, as claimed. This completes the argument. □

F.1. Construction for Minimum Non-Negative Sum (MNNS) Task

We describe a single-layer transformer with an attention block followed by a mixture-of-experts (MoE) feed-forward block. Let n be the length of the input sequence of integer tokens. Denote the tokenized input numbers as $z_1, z_2, \ldots, z_n$; and let the arrow ($\rightarrow$) token be denoted as z_{n+1}. We also have a dummy input token z_{n+2}, which is the embedding corresponding to number 0, so that we have $n+2$ tokens initially. We will construct the transformer with $n+1$ MLPs in the mixture of experts layer, where the first n are partial-sum MLPs and the last one is the MLP that reads off the answer from among the all stored partial sums after m steps. We start with the following assumption on the structure of the tokens.

Assumption 2. *Let $d = d_e + d_p$ be the embedding size where $d_e = 2^{n+1}$ and $d_p = n+2$. The token embeddings are on the first d_e coordinates, while the positional encodings are on the last d_p coordinates and are one-hot encoded. where each $z_i = \begin{pmatrix} \boldsymbol{e}_i \\ \mathbf{p}_i \end{pmatrix} \in \mathbb{R}^{d_e+d_p}$ is formed by vertically concatenating a content embedding $\boldsymbol{e}_i \in \mathbb{R}^{d_e}$ and a positional encoding $\mathbf{p}_i \in \mathbb{R}^{d_p}$. We assume each $\boldsymbol{p}_i$ is a one-hot vector in $\mathbb{R}^{d_p}$, so that $\boldsymbol{p}_i^\top \boldsymbol{p}_j = 0$ for $i \neq j$, and $\|\mathbf{p}_i\| = 1$.*

We now state the following proposition, which helps us to attend and select the input tokens $z_1, \ldots, z_{n+1}$ one by one by the attention block.

Proposition 4. *Suppose we have $n+2$ tokens $\{z_1, z_2, \ldots, z_{n+2}\}$ in $\mathbb{R}^d$, each of the form $z_i = \begin{pmatrix} \boldsymbol{e}_i \\ \boldsymbol{p}_i \end{pmatrix}$, where $\boldsymbol{e}_i \in \mathbb{R}^{d_e}, \boldsymbol{p}_i \in \mathbb{R}^{d_p}, d = d_e + d_p$. Let $p_1, p_2, \ldots, p_{n+2} \in \mathbb{R}^{d_p}$ be orthonormal set of positional vectors according to Assumption 2. Then, there exists a rotation matrix $\boldsymbol{R} \in \mathbb{R}^{d_p \times d_p}$ satisfying $\boldsymbol{R}\boldsymbol{p}_j = \boldsymbol{p}_{j-1 \bmod (n+2)}$ for all $j \in [n+2]$, and the block matrices*

$$\boldsymbol{W} = \begin{pmatrix} \mathbf{0}_{d_e \times d_e} & \mathbf{0}_{d_e \times d_p} \\ \mathbf{0}_{d_p \times d_e} & c \cdot \boldsymbol{R} \end{pmatrix} \in \mathbb{R}^{d \times d} \quad \textit{and} \quad \boldsymbol{W}_v = \begin{pmatrix} I_{d_e} & \mathbf{0}_{d_e \times d_p} \\ \mathbf{0}_{d_p \times d_e} & \boldsymbol{I}_{d_p} \end{pmatrix} \in \mathbb{R}^{d \times d}$$

with $c \rightarrow \infty$, ensure that the attention block

$$\mathrm{Attn}(z, \boldsymbol{Z}) = \mathbb{S}\left(z^\top \boldsymbol{W} \boldsymbol{Z}^\top\right) \boldsymbol{Z} \boldsymbol{W}_v,$$

performs a cyclic next-index selection: if the query is z_i, it selects column $j^* \equiv (i+1) \pmod{n+2}$ from $\boldsymbol{Z}$ and returns z_{j^*}.

Proof. Definition of Matrix $\boldsymbol{W}$. We will first construct a rotation matrix. We have $n+2$ orthonormal position vectors $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{n+2} \in \mathbb{R}^{d_p}$. Then, $\boldsymbol{R}$ is the following $(n+2) \times (n+2)$ permutation matrix

$$\boldsymbol{R} = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix},$$

which cyclically shifts the basis vectors $\boldsymbol{p}_j$ backward by one index, i.e., $\boldsymbol{R}\boldsymbol{p}_j = \boldsymbol{p}_{j-1 \bmod (n+2)}$. Then, we specify

$$\boldsymbol{W} = \begin{pmatrix} \boldsymbol{0}_{d_e \times d_e} & \boldsymbol{0}_{d_e \times d_p} \\ \boldsymbol{0}_{d_p \times d_e} & \boldsymbol{R} \end{pmatrix} \in \mathbb{R}^{d \times d}.$$

Hence for $\boldsymbol{z}_i = (\boldsymbol{e}_i; \boldsymbol{p}_i)$, we have $\left(\boldsymbol{e}_i^\top, \boldsymbol{p}_i^\top\right)\boldsymbol{W} = \left(\boldsymbol{0}, \boldsymbol{p}_i^\top \boldsymbol{R}\right)$. Thus the dot-product with $\boldsymbol{z}_j$ is

$$\left(0 \ \boldsymbol{p}_i^\top \boldsymbol{R}\right)\begin{pmatrix} \boldsymbol{e}_j \\ \boldsymbol{p}_j \end{pmatrix} = \boldsymbol{p}_i^\top \boldsymbol{R}\boldsymbol{p}_j.$$

Since positional encodings are orthogonal, we know that:

$$\boldsymbol{p}_i^\top \boldsymbol{R}\boldsymbol{p}_j = \begin{cases} 1, & j \equiv i + 1 (\mathrm{mod}(n+2)), \\ 0, & \text{else.} \end{cases}$$

So row-wise softmax $\mathbb{S}\left(\boldsymbol{x}^\top \boldsymbol{W}\boldsymbol{X}^\top\right)$ places all probability mass at column $j^* \equiv i + 1(\mathrm{mod}(n+2))$ by saturating softmax at position j as $c \to \infty$.

Definition of Matrix $\boldsymbol{W}_v$. In this case, we simply set $\boldsymbol{W}_v = \boldsymbol{I}_d$, and thus, once the row-wise softmax selects column j^* with probability 1, we have

$$\boldsymbol{z}_{j^*}^\top \boldsymbol{W}_v = \boldsymbol{z}_{j^*},$$

so the final output is precisely the chosen $\boldsymbol{z}_{j^*}$. This completes the construction. □

Having defined the attention block, we state the following proposition that helps selecting different MLPs for the tokens $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{n+1}$ outputted by the attention block.

Having defined the attention block, we now show how a mixture-of-experts layer can exclusively select MLP_i for each token $\boldsymbol{z}_i$, $i = 1, \ldots, n+1$ outputted by the attention block.

Proposition 5. *Let $MLP_1, \ldots, MLP_{n+1}$ be $n+1$ experts in a mixture-of-experts (MoE) module. Suppose we have $n+1$ fixed token embeddings $\{z_1, z_2, \ldots, z_{n+1}\} \subset \mathbb{R}^d$, where each token is formed according to Assumption 2. Given routing parameters $\boldsymbol{W} = [\boldsymbol{w}_1 \ \ldots \ \boldsymbol{w}_{n+1}]^\top$, define the MoE feed-forward block as*

$$\mathrm{MoEBlock}(\boldsymbol{z}) = \sum_{j=1}^{n+1} \left[\mathrm{Softmax}(\mathrm{Wz})_j \cdot MLP_j(\boldsymbol{z})\right],$$

where

$$\mathrm{Softmax}(\boldsymbol{W}\boldsymbol{z})_j = \frac{\exp\left(\boldsymbol{w}_j^\top \boldsymbol{z}\right)}{\sum_{k=1}^{n+1} \exp\left(\boldsymbol{w}_k^\top \boldsymbol{z}\right)}, \quad j = 1, \ldots, n+1.$$

There exist routing matrix $\boldsymbol{W} \in \mathbb{R}^{(n+1)\times d}$ such that the distribution $\mathrm{Softmax}(c \cdot \boldsymbol{W}\boldsymbol{z}_i)$ as $c \to \infty$ assigns a weight of 1 on MLP_i when $\boldsymbol{z}_i$ is given as input.

Proof. We partition $\boldsymbol{w}_j$ to ignore the content embedding $\boldsymbol{e}_i$ and match the positional block $\boldsymbol{p}_j$. Concretely, write $\boldsymbol{w}_j = \begin{pmatrix} \boldsymbol{0}_{d_e} \\ \boldsymbol{p}_j \end{pmatrix}$. Then, for each token $\boldsymbol{z}_i = (\boldsymbol{e}_i; \boldsymbol{p}_i)$,

$$\boldsymbol{w}_j^\top \boldsymbol{z}_i = \begin{pmatrix} \boldsymbol{0}_{d_e}^\top & \boldsymbol{p}_j^\top \end{pmatrix}\begin{pmatrix} \boldsymbol{e}_i \\ \boldsymbol{p}_i \end{pmatrix} = \boldsymbol{p}_j^\top \boldsymbol{p}_i.$$

Since $\boldsymbol{p}_j^\top \boldsymbol{p}_i = \delta_{ij}$, we have $\boldsymbol{w}_j^\top \boldsymbol{z}_i = \delta_{ij}$. Therefore, the softmax evaluates to

$$\lim_{c \to \infty} \mathrm{Softmax}(c \cdot \boldsymbol{W}\boldsymbol{z}_i)_j \to \frac{\exp(\delta_{ij})}{\sum_{k=1}^{n+1} \exp(\delta_{ik})} = \delta_{ij}.$$

In other words, $\mathrm{Softmax}(c \cdot \boldsymbol{W}\boldsymbol{z}_i)$ places all mass on expert $j = i$. Thus each token $\boldsymbol{z}_i$ (for $i = 1, \ldots, n+1$) deterministically selects the i-th expert MLP_i. □

In the next proposition, we show how to iteratively expand the partial sums by adding and subtracting the digit obtained from the attention block and write each resulting sum to a distinct spot in the output vector.

Proposition 6 (Partial-Sum MLPs)**.** *Suppose that the embedding dimension d satisfies $d \geq 2^{j+1} + d_p$. Let z_{prev} contain the 2^{j-1} partial sums s_k each encoded by a pair $(\cos(\omega s_k),\ \sin(\omega s_k))$ of coordinates such that:*

$$z_{prev} = \left[\cos(\omega s_1)\ \sin(\omega s_1)\ \ldots\ \cos(\omega s_{2^{j-1}})\ \sin(\omega s_{2^{j-1}})\, 0 \ldots 0\right]^\top \in \mathbb{R}^d,$$

and let z_{curr} contain the input digit d_j encoded in the first two coordinates:

$$z_{curr} = \left[\cos(\omega d_j)\ \sin(\omega d_j)\, 0\ \ldots\ \ 0\right]^\top \in \mathbb{R}^d.$$

Then, for any $1 \leq j \leq n$, there exist $MLP_j : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ such that when (z_{prev}, z_{curr}) is given as input, it outputs the vector $z_{out} \in \mathbb{R}^d$ so that its first 2^j coordinate-pairs store the trigonometric encodings of $(s_k + d_j)$, and the next 2^j coordinate-pairs store those of $(s_k - d_j)$. Formally, first 2^j coordinates are $[\cos(\omega(s_k + d_j)), \sin(\omega(s_k + d_j))]$ for all partial sums s_k, and the next 2^j coordinates are $[\cos(\omega(s_k - d_j)), \sin(\omega(s_k - d_j))]$ for all partial sums s_k, with any remaining coordinates set to zero.

Proof. Each expert MLP_j (for $1 \leq j \leq n$) adds j-th integer d_j in both its positive and negative form to all previously computed partial sums. For simplicity, let's say that j-th integer to add is d_j. By trigonometric identities, we know that

$$\cos(\omega(s_k + d_j)) = \cos(\omega s_k)\cos(\omega d_j) - \sin(\omega s_k)\sin(\omega d_j),$$
$$\sin(\omega(s_k + d_j)) = \sin(\omega s_k)\cos(\omega d_j) + \cos(\omega s_k)\sin(\omega d_j),$$

and similarly,

$$\cos(\omega(s_k - d_j)) = \cos(\omega s_k)\cos(\omega d_j) + \sin(\omega s_k)\sin(\omega d_j),$$
$$\sin(\omega(s_k - d_j)) = \sin(\omega s_k)\cos(\omega d_j) - \cos(\omega s_k)\sin(\omega d_j).$$

Using the above identities, we will obtain the sum by introducing matrices that do shift/swap operations. Concretely, for $k = 1, \ldots, 2^m$, the k-th 2×2 block acts on $\begin{pmatrix}\cos(\omega s_k)\\ \sin(\omega s_k)\end{pmatrix}$ in z_{prev}. We define:

$$\boldsymbol{W}^+_{\sin} = \operatorname{diag}\left(\underbrace{\begin{pmatrix}0 & -1\\ 1 & 0\end{pmatrix}, \ldots, \begin{pmatrix}0 & -1\\ 1 & 0\end{pmatrix}}_{2^{j-1}\text{ blocks}}, 0, \ldots, 0\right),$$

$$\boldsymbol{W}^-_{\sin} = \operatorname{diag}\left(\underbrace{\begin{pmatrix}0 & 1\\ -1 & 0\end{pmatrix}, \ldots, \begin{pmatrix}0 & 1\\ -1 & 0\end{pmatrix}}_{2^{j-1}\text{ blocks}}, 0, \ldots, 0\right).$$

The above constructions of $\boldsymbol{W}^+_{\sin}$ and $\boldsymbol{W}^-_{\sin}$ satisfy,

$$\boldsymbol{W}^+_{\sin} z_{\text{prev}} = \left[-\sin(\omega s_1)\ \cos(\omega s_1) \cdots -\sin(\omega s_{2^{j-1}})\ \cos(\omega s_{2^{j-1}})\, 0 \ldots 0\right]^\top \in \mathbb{R}^d$$

and

$$\boldsymbol{W}^-_{\sin} z_{\text{prev}} = \left[\sin(\omega s_1)\ -\cos(\omega s_1) \ldots \sin(\omega s_{2^{j-1}})\ -\cos(\omega s_{2^{j-1}})\, 0 \ldots 0\right]^\top \in \mathbb{R}^d.$$

Each of these act blockwise on the first 2^j coordinates of z_{prev} and zeroes out everything else in dimension d. We also have $z_{\text{curr}} \in \mathbb{R}^d$ with two designated coordinates $z_{\text{curr},1} = \cos(\omega d_j)$, and $z_{\text{curr},2} = \sin(\omega d_j)$, with all other coordinates being zero. We multiply z_{prev} by $\cos(\omega d_j)$ and $\sin(\omega d_j)$ elementwise. Formally, the sum

$$z_{\text{curr},1} \cdot z_{\text{prev}} + z_{\text{curr},2} \cdot (M^+_{\sin}\, z_{\text{prev}})$$

gives the 2^{j-1} partial sums $\{s_k + d_j\}_{k=1}^{2^{j-1}}$ stored in the coordinates from 1 to 2^j. We define $\boldsymbol{W}_{\text{shift}} \in \mathbb{R}^{d\times d}$ in a block form with three row blocks and two column blocks:

$$\boldsymbol{W}_{\text{shift}} = \begin{pmatrix} \boldsymbol{0}_{2^j\times 2^j} & \boldsymbol{0}_{2^j\times(d-2^j)} \\ \boldsymbol{I}_{2^j} & \boldsymbol{0}_{2^j\times(d-2^j)} \\ \boldsymbol{0}_{(d-2^{j+1})\times 2^j} & \boldsymbol{0}_{(d-2^{j+1})\times(d-2^j)} \end{pmatrix}.$$

When applied, the above matrix shifts the first 2^j entries of $\boldsymbol{z}_{\text{prev}}$ by 2^j coordinates. Now, also define

$$\boldsymbol{z}_{\text{curr},2} \cdot \left(\boldsymbol{W}_{\text{shift}} \boldsymbol{W}_{\text{sin}}^{-} \boldsymbol{z}_{\text{prev}}\right) + \boldsymbol{z}_{\text{curr},1} \cdot \left(\boldsymbol{W}_{\text{shift}} \boldsymbol{z}_{\text{prev}}\right).$$

This way, the above sum gives us the 2^{j-1} partial sums $\{s_k - d_j\}_{k=1}^{2^{j-1}}$ stored in the coordinates from $2^j + 1$ to 2^{j+1} encoded in trigonometric format. Then, we normalize this output of the model by $1/2$ and obtain the following output:

$$\begin{aligned}
&\frac{1}{2}\left(\boldsymbol{z}_{\text{curr},1} \cdot \boldsymbol{z}_{\text{old}} + \boldsymbol{z}_{\text{curr},2} \cdot \left(M_{\text{sin}}^{+} \boldsymbol{z}_{\text{old}}\right) + \boldsymbol{z}_{\text{curr},2} \cdot \left(\boldsymbol{W}_{\text{shift}} \boldsymbol{W}_{\text{sin}}^{-} \boldsymbol{z}_{\text{prev}}\right) + \boldsymbol{z}_{\text{curr},1} \cdot \left(\boldsymbol{W}_{\text{shift}} \boldsymbol{z}_{\text{prev}}\right)\right) \\
&= \frac{1}{2}\Big[\cos\left(\omega\left(s_1 + d_j\right)\right), \sin\left(\omega\left(s_1 + d_j\right)\right), \ldots, \cos\left(\omega\left(s_{2^{j-1}} + d_j\right)\right), \sin\left(\omega\left(s_{2^{j-1}} + d_j\right)\right), \\
&\quad \cos\left(\omega\left(s_1 - d_j\right)\right), \sin\left(\omega\left(s_1 - d_j\right)\right), \ldots, \cos\left(\omega\left(s_{2^{j-1}} - d_j\right)\right), \sin\left(\omega\left(s_{2^{j-1}} - d_j\right)\right), \\
&\quad 0, \ldots, 0\Big]^{\top} \in \mathbb{R}^d.
\end{aligned}$$

Thus, this is exactly the representation of 2^j partial sums. This completes the argument. We should remark that, the above argument utilizes a *gated MLP* which explicitly multiplies the elements of the input features, namely, $\boldsymbol{z}_{\text{curr}}$ with the partial sums $\boldsymbol{z}_{\text{prev}}$. On the other hand, we don't require any nonlinear activation function, so our MLP constructions have the form $\text{MLP}(\boldsymbol{z}) = \boldsymbol{W}_3(\boldsymbol{W}_1\boldsymbol{z} \odot \boldsymbol{W}_2\boldsymbol{z})$ for suitable choices of $\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3$ where $\odot$ denotes the Hadamard product. The use of gated MLPs is a standard practice in transformer architectures (Shazeer, 2020). □

Proposition 7 (Read-Off MLP)**.** *Suppose that every partial sum s_k is in the range $[-S, S]$ and let $\omega < \pi/2S$. Assume that the vector*

$$\boldsymbol{z} = [\cos(\omega s_1),\ \sin(\omega s_1),\ \ldots,\ \cos(\omega s_{2^n}),\ \sin(\omega s_{2^n}), 0, \ldots, 0]^{\top} \in \mathbb{R}^d,$$

contains 2^n partial sums $\{s_1, \ldots, s_{2^n}\}$ encoded in trigonometric form, where $d = 2^{n+1} + n + 2$. Then there exists a single feed-forward network $MLP_{n+1} : \mathbb{R}^d \to \mathbb{R}^d$ such that, given input $\boldsymbol{z}$, it selects the smallest nonnegative s_ℓ from $\{s_1, \ldots, s_{2^n}\}$ and outputs the embedding $e_{s_\ell} \in \mathbb{R}^d$, where s_ℓ is that minimal nonnegative partial sum.

Remark: Our construction relies on gated MLP, rather than standard MLP, as in Proposition 6.

Proof. We know that the input embedding $\boldsymbol{z}$ represents 2^n pairs, each pair $(\cos(\omega s_i),\ \sin(\omega s_i))$ stored consecutively. That is,

$$\boldsymbol{z} = [\cos(\omega s_1),\ \sin(\omega s_1),\ \ldots,\ \cos(\omega s_{2^n}),\ \sin(\omega s_{2^n}), 0, \ldots, 0]^{\top} \in \mathbb{R}^d,$$

We will identify the smallest $s_\ell \geq 0$ and output an embedding e_{s_ℓ} denoting that integer. We are given that ω is small enough such that when $s_\ell \in [0, S]$, we ensure $S\omega < \pi/2$. This guarantees $\sin(\omega s_\ell) \geq 0$ if and only if $s_\ell \geq 0$. First, we wish to collapse $\boldsymbol{z}$ into a single vector of size 2^n, keeping $\cos(\omega s_\ell)$ only when $\sin(\omega s_\ell) \geq 0$ and zeroing it out otherwise. We define two matrices $\boldsymbol{W}_{\cos}, \boldsymbol{W}_{\sin} \in \mathbb{R}^{d\times d}$ by

$$\begin{aligned}
(\boldsymbol{W}_{\cos})_{i,(2i-1)} &= 1, \quad (\boldsymbol{W}_{\cos})_{i,j} = 0 \quad \text{for } j \neq 2i-1, \\
(\boldsymbol{W}_{\sin})_{i,(2i)} &= 1, \quad (\boldsymbol{W}_{\sin})_{i,j} = 0 \quad \text{for } j \neq 2i.
\end{aligned}$$

for $1 \leq i \leq 2^n$ and all other rows/columns of $\boldsymbol{W}_{\sin}, \boldsymbol{W}_{\cos}$ are zero. Hence each matrix picks out alternate coordinates:

$$\boldsymbol{z}_{\cos} = \boldsymbol{W}_{\cos}\,\boldsymbol{z} = \begin{bmatrix} \cos(\omega s_1) \\ \cos(\omega s_2) \\ \vdots \\ \cos(\omega s_{2^n}) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^d, \quad \boldsymbol{z}_{\sin} = \boldsymbol{W}_{\sin}\,\boldsymbol{z} = \begin{bmatrix} \sin(\omega s_1) \\ \sin(\omega s_2) \\ \vdots \\ \sin(\omega s_{2^n}) \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^d.$$

In order to find the minimum non-negative number, we need to find the number s such that it maximizes $\cos(\omega s)$ and satisfies $\sin(\omega s) \geq 0$. For this, we utilize a sigmoid activation function in the following way:

$$z_{\text{filter}} = z_{\cos} \odot \sigma\left(c\, z_{\sin}\right),$$

where $\sigma(x) = \frac{1}{1+\exp(-x)}$ is element-wise sigmoid function, and $c \to \infty$ is a large constant. With this choice of c, the sigmoid output will be 1 when $s_\ell \geq 0$ and 0 otherwise. Therefore, the resulting vector z_{filter} contains $\cos(\omega s)$ values at indices where $\sin(\omega s)$ is positive. Now, for $0 \leq s_\ell \leq S$ with $S\,\omega \leq \frac{\pi}{2}$, the ordering of s_ℓ from smallest to largest is the same as the ordering of $\cos(\omega s_\ell)$ from largest to smallest. Thus, to find the minimum nonnegative sum, we find the partial sum ℓ^* that maximizes $\cos(\omega s_\ell)$. Utilizing another gating, we calculate

$$\text{softmax}\left(c\, z_{\text{filter}}\right)^\top z_{\text{filter}}$$

as $c \to \infty$. The softmax vector will be one-hot with 1 at index ℓ^* that has the largest $\cos(\omega s_\ell)$. A second multiplication with z_{filter} will return this $\cos(\omega s_{\ell^*})$. Therefore, $\text{softmax}\left(c\, z_{\text{filter}}\right)^\top z_{\text{filter}} = \cos(\omega s_{\ell^*})$. Next, we retrieve the corresponding sine entry of s_{ℓ^*} by applying the same one-hot selection to $z_{\sin}$. Formally,

$$\text{softmax}\left(c\, z_{\text{filter}}\right)^\top z_{\sin} = \sin\left(\omega s_{\ell^*}\right),$$

as $c \to \infty$. Hence, from these two selected coordinates, $[\cos(\omega s_{\ell^*}), \sin(\omega s_{\ell^*})]$, we produce the final embedding in $\mathbb{R}^d$ by placing them in the first two coordinates and zeros elsewhere:

$$\boldsymbol{e}_{s_{\ell^*}} = [\cos(\omega s_{\ell^*}),\ \sin(\omega s_{\ell^*}),\ 0, \ldots, 0]^\top,$$

where s_{ℓ^*} is the minimal nonnegative sum. This completes the argument. □

Proposition 1. *There exists a* 1*-layer transformer architecture with a mixture-of-experts MLP layer that solves the MNNS task using CoT2 and following the CSFT supervision in Section 3.*

Proof. We will argue that by combining Propositions 5 to 7, we obtain a single-layer transformer that is formed by an attention block followed by an MoE feed-forward block, which solves the Minimum Non-Negative Sum (MNNS) task.

Suppose that we have n input integers $d_1, \ldots, d_n$, encoded as $z_1, \ldots, z_n$, plus an arrow ($\rightarrow$) token z_{n+1} and a dummy token z_{n+2} corresponding to the integer 0. In this case, we will output the tokens representing the ground-truth sums $s_1, \ldots, s_n$, therefore, the number of output tokens is $m = n$ in MNNS setting. We assume that the inputs are encoded according to Assumption 2. By Proposition 6, there exist $\text{MLP}_1, \ldots, \text{MLP}_n$ that perform the following: whenever MLP_j is selected with input $(z_{\text{prev}}, z_{\text{curr}})$ such that z_{prev} stores 2^{j-1} partial sums and z_{curr} stores the digit d_j, it adds and subtracts d_j to all previously stored partial sums and stores the resulting 2^j partial sums in z_{out}. The dummy token z_{n+2} that corresponds to the integer 0 allows us to initialize the partial sums from zero. If the query token is z_{n+2}, we produce the first partial sums by combining this dummy 0 with d_1, which are $(+d_1)$ and $(-d_1)$ encoded in an output token.

We assign positional encodings cyclically to output tokens. That means, the first $n+2$ input tokens have positional encodings from $\boldsymbol{p}_1$ to $\boldsymbol{p}_{n+2}$, and the output tokens have $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots,$ as their positional encodings, in this exact order. This way, by Proposition 4, $\text{Attn}(z, \boldsymbol{Z})$ attends and selects the input digit tokens $z_1, z_2, \ldots, z_n$ and finally arrow z_{n+1} one by one and feeds to $\text{MoEBlock}(\cdot)$.

By Proposition 5, there's a $\text{MoEBlock}(z)$ such that if the input is z_j (for $j \leq n$), MLP_j is selected with probability 1, and if the input is arrow token z_{n+1}, MLP_{n+1} is selected with probability 1, which is the MLP to read-off the final answer. In the input tokens $z_1, \ldots, z_n$, the first two coordinates store the trigonometric representation of $d_1, \ldots, d_n$. To allow outputting the final answer by MLP_{n+1}, the partial sums obtained in the intermediate steps need to be written to separate coordinates. Therefore, MLP_j takes a vector filled in the first 2^j coordinates, adds d_j and writes to the first 2^j coordinates, subtracts d_j and writes to the next 2^j coordinates, and finally divides the entire representation by 2 to maintain consistent scaling since the number of partial sums is doubled. In other words, the first n MLPs have some repeated behavior. Finally, by Proposition 7, MLP_{n+1} receives a vector that encodes all 2^n possible partial sums in cos/sin form in 2^{n+1} coordinates and extracts the embedding of smallest nonnegative number among them.

Altogether, this single-layer transformer with an attention module to pass the tokens to the mixture-of-experts MLP solves the Minimum Non-Negative Sum task by following CSFT described in 3. □

G. Implementation Details

Computational Resources: All experiments were run on a Slurm-managed cluster using L40S GPUs with 48GB of memory. Each experiment fits on a single GPU. In the case of 4 input digits, the SFT or CSFT training takes approximately 3 hours on a single GPU. For 5-digit inputs, the dataset size increases by roughly a factor of 10, and the training time increases proportionally. The entire codebase was implemented in PyTorch.

G.1. Implementation Details of Experiments on MNNS Task

Dataset Details: For the MNNS task, the vocabulary consists of a range of numbers from $[-S, S]$ for some positive integer S, together with <*BOS*>, <*EOS*>, and $\rightarrow$ special tokens. The integer S is chosen so that all possible partial sums of the selected input digits lie within $[-S, S]$. For example, when the input digits lie in the range 1–10, we set $S = 36$, whereas for digits in 5–14, we set $S = 40$. We performed our experiments on the 4 and 5 input digit scenarios. A sample input line with m numbers is:

$$<BOS> D_1 D_2 \ldots D_m \rightarrow$$

Accordingly, the output will be m sum tokens, where the final token corresponds to the answer, followed by <*EOS*> token:

$$S_1 S_2 \ldots S_m <EOS>$$

As a concrete example, consider the input $2, 1, 4$ ($m = 3$), following Figure 1. In this case, the solution for the MNNS task is $-2 - 1 + 4 = 1$. Therefore, for the discrete model, the input is <*BOS*> $D_2 D_1 D_4 \rightarrow$ and we supervise it along the trajectory of correct output tokens $S_{-2} S_{-3}, S_1$<*EOS*>, as illustrated in Figure 1. On the other hand, the continuous supervision at the first step holds S_2 and S_{-2} as possibilities. Then, for the next step, we add 1 or -1 to these numbers, and the resulting possibilities are S_3, S_1, S_{-1}, S_{-3}. Finally, at the last step, the model is supervised to pick the correct answer S_1 as the token.

We split the datasets by ensuring that each permutation of a set of numbers is exactly in one of the train and validation datasets, as the answer to the question is permutation-invariant. This way, we prevent the models from memorizing the answer and make a fair comparison. We also use 0.8-0.2 split for train-val datasets.

Model and Hyperparameters: We use the GPT2 model, with 1 layer 1 head, 2 layer 2 head, and 4 layer 4 head as the configurations. For each configuration, we experiment with embedding dimensions of 16, 24, or 32. We train with a learning rate of `lr` $= 10^{-4}$ and use AdamW (no weight decay). The batch size is 16 for 4-digit inputs and 64 for 5-digit inputs.

Evaluation of the models: To make a proper comparison, we only check the final answer of the models, as checking the correctness of the full path of the discrete model would be unfair.

Pass@k Experiment: We perform our experiments for temperatures 0, 0.4, 0.8, and 1 by repeating the evaluation 10 times for each k value where k changes from 1 to 14.

G.2. Implementation Details of Experiments on ProntoQA/ProsQA Datasets

Dataset Details: Different from the original ProntoQA/ProsQA datasets which described the structured ontology in natural language as a set of known conditions, we use a more structured format through a token-level representation. An example prompt is shown below.

Description of the structured ontology: Each component of the ontology and associated questions is represented through discrete tokens with their own learned embeddings, rather than as raw textual input. Specifically, we use the GPT-2 architecture and encode the ontology's structural components. Below are two examples demonstrating how natural-language assertions are mapped to our tokenized format:

`Brimpuses are not luminous` $\rightarrow$ `'A' 'not in' 'B' '.'`.

`Shumpuses are amenable; Each yumpus is a lorpu; Every lorpus is floral` $\rightarrow$ `'C' 'in' 'D' '.'`.

Below, we have the ProntoQA and ProsQA datasets' input-output format.

The structure of ProntoQA:

Input:`'Description' '{' 'A' 'not in' 'B' '.' ... 'C' 'in' 'D' '.' '}' 'Question' '{' 'C' 'not in' 'F' '.' '}'`

Output:`'Steps' { 'C' 'in' 'D' '.' ... 'D' 'in' 'E' '.' '}' 'Answer' '{' 'False' '}'`

The structure of ProsQA:

Input:`'Description' '{' 'A' 'in' 'B' '.' ... 'C' 'in' 'D' '.' '}' 'Question' '{' 'C' 'in' 'F' 'or' 'E' '}'`

Output:`'Steps' { 'C' 'in' 'D' '.' ... 'D' in 'E' '.' '}' 'Answer' '{' 'F' '}'`

Each distinct component or relation (e.g., `'A'`, `'in'`, `'not in'`) is treated as a unique token, and singular/plural variants (such as `'lempus'` and `'lempuses'`) are collapsed into a single token to simplify the vocabulary. Alongside these concept tokens, special structural tokens (`'Description'`, `'', ''`, `'.'`, `'or'`, etc.) are also included, which results in a vocabulary size of 31 tokens. To avoid biases, we balance the dataset. In ProntoQA, "yes" and "no" each appear with 50% probability, and in ProsQA, the correct answer is randomly permuted at the first or second position. For all the other experimental and training settings, we follow (Hao et al., 2024).

Model and Hyperparameters: We use the GPT2 model, with 2 layer 2 head, and 4 layer 4 head as the configurations. We tested embedding dimensions $24, 32, 40$ with these configurations. We set batch size 64. We train with a learning rate of 10^{-4} and use AdamW (no weight decay).

Maj@k Experiment: We use majority voting for evaluation instead of Pass@k, because both ProntoQA and ProsQA are binary questions. We perform our experiments for temperatures 0, 0.4, 0.8, and 1 by repeating the evaluation 10 times for each k value where k changes from 1 to 21. If two or more answers end up with the same top vote, we pick one randomly.

G.3. Implementation Details of GRPO Training

In (Hao et al., 2024), the reference model is updated by $\text{LM}_{\theta_{\text{ref}}} \leftarrow \text{LM}_\theta$ in each iteration (epoch). This approach is reasonable for their setting with a large dataset and a small number of epochs over it. For our setting, however, we set the reference model to the initial model and never update it through iterations as we have a smaller dataset. Meanwhile, we update the old model before every batch $\text{LM}_{\theta_{\text{old}}} \leftarrow \text{LM}_\theta$.

In our experiments, we use $G = 8$ trajectories per input data point, use clipping parameter $\epsilon = 0.1$, and set the KL-divergence coefficient $\beta = 0$ in most cases (with $\beta = 0.1$ in a few). For the CoT2 model with MTS sampling, we change the number of tokens to sample K from 1 to 12. In the MNNS task, the 5-digit case has about ten times more data than the 4-digit case, so we typically focus on 4-digit MNNS because of computational considerations and use a batch size of 16 in those experiments.

Learning rates differ by model and setting. We use $\texttt{lr} = 5 \times 10^{-5}$ for CoT2 + MTS sampling (figures in the main text), $\texttt{lr} = 1 \times 10^{-5}$ for discrete CoT with Dirichlet sampling, and $\texttt{lr} = 1 \times 10^{-6}$ for CoT2 with Dirichlet sampling. For ProntoQA and ProsQA experiments, we perform a grid search over learning rates ranging from 1×10^{-4} to 1×10^{-8} and select and report results using the best-performing configuration. For most settings, we find $\texttt{lr} = 1 \times 10^{-5}$ optimal; however, for CoT2 and discrete CoT models with $K = 6$, we set $\texttt{lr} = 1 \times 10^{-6}$.We also use AdamW with a weight-decay of 0.01. For Dirichlet experiments on the MNNS task, we try various scale parameters γ, but we find $\gamma = 20$ to work best in most settings. Unless stated otherwise, we report the best validation accuracy found during training for each setting.