CAN THE FUTURE INFORM THE PRESENT? INVESTIGATING LATENT LOOKAHEAD REFINEMENT VIA MULTI-TOKEN PREDICTION

Somesh Mehra, Alejandro Hernández-Cano, & Martin Jaggi
EPFL
somesh.mehra@epfl.ch

ABSTRACT

Most modern language models operate within the autoregressive paradigm, and are trained to perform next-token prediction (NTP) conditioned only on the previous context. However, this results in myopic generation, since these models have no future information to help guide their prediction, which limits their effectiveness in tasks that require lookahead. Recently, multi-token prediction (MTP) has shown promise as an auxiliary training objective to learn more future-aware model parameters, however existing methods typically discard MTP at inference time, or use it only for speculative decoding. In this work, we propose to additionally leverage the future-token prediction capability of MTP at inference for latent lookahead refinement, moving from implicitly future-aware to explicitly future-informed token generation. Our method introduces an internal latent thinking mechanism that operates within a single forward pass and incurs minimal inference overhead. Whilst early results are inconclusive, our experiments suggest that the proposed method could prove to be more effective at scale, and we believe our analyses can serve as foundation for future work.

1 INTRODUCTION

Autoregressive models, which remain the dominant paradigm in modern language modeling, are typically trained for next-token prediction (NTP) and operate in a strictly causal, left-to-right fashion. During generation, each new token is conditioned exclusively on the preceding context, meaning that there is no explicit representation of what is coming in the future to help inform the decision of which token to predict. This results in myopic generation, since the model is effectively blind to the downstream consequences of its current prediction.

For many language modeling tasks however, knowing what is coming in the future can be helpful, or sometimes even crucial to decipher the optimal current token. For example, Baeumel et al. (2025) show that LLMs consistently fail on the relatively simple task of multi-operand addition due to a one-token lookahead not being able to accurately factor cascading carries. Bachmann & Nagarajan (2024) and Nagarajan et al. (2025) further demonstrate that NTP is myopic and can fail in tasks that require lookahead. Additionally, if a model does not consider the future, it is more susceptible to falling into suboptimal paths which may even make correctly predicting the remaining sequence impossible without some form of backtracking.

Numerous works have aimed to address these problems by inducing planning capabilities in LLMs. One common direction is to train these models with an additional future prediction objective to augment the myopic NTP objective, with the aim of learning better representations that allow longer-term planning. For example, methods such as multi-token prediction (MTP) (Gloeckle et al., 2024; Gerontopoulos et al., 2025) and future summary prediction (Mahajan et al., 2025) have been shown to be effective in improving LLM performance on tasks which require lookahead (Nagarajan et al., 2025). Yet, a critical gap remains: these methods typically only use future prediction as an auxiliary training signal, but at inference time, generation falls back to being conditioned only on the preceding context without any explicit future information.

In this work, we propose a novel internal latent thinking mechanism to leverage this future prediction capability at inference time rather than discarding it, which we call LaLoR: Latent Lookahead Refinement using MTP. More concretely, our method builds upon recent token-based MTP methods (Gerontopoulos et al., 2025) and splits the model into two stages: 1) a *draft* stage, where the model projects a future trajectory in parallel to the current token prediction; and 2) a *refine* stage, where the current token is allowed to look ahead at the projected future in latent space to refine its prediction accordingly (achieved by simply modifying the attention mask). Thus, we introduce a per-token thinking mechanism which occurs *within a single forward pass*, and incurs minimal inference overhead. With our method, we aim to achieve a future-informed inference process to enable non-myopic generation.

Our empirical evaluation focuses on the continual pretraining setting. While early results suggest that an MTP training objective may be beneficial over standard NTP, the proposed lookahead mechanism shows limited benefit compared to the MTP baseline. We offer a range of hypotheses and experiments to explain this behaviour. In particular, we believe that currently, the small models we use do not learn strong enough future latent representations to aid the next-token generation process, and larger scale experiments (both in terms of data budget and model parameter count) could provide for a fairer opportunity to assess the potential benefits of our lookahead mechanism.

2 Related works

Multi-Token Prediction Many recent works explore MTP, mostly for two key uses: 1) as an additional training objective to augment the myopic NTP objective (Gloeckle et al., 2024; Liu et al., 2024; Gerontopoulos et al., 2025), and/or 2) for self-speculative decoding for faster inference (Cai et al., 2024; Chen et al., 2024; Samragh et al., 2025). MTP is commonly achieved by appending additional prediction heads atop a shared backbone (Gloeckle et al., 2024; Liu et al., 2024; 2025b), or by introducing special tokens which perform a future-token prediction (Gerontopoulos et al., 2025). Regardless of the method however, as far as we know, all prior MTP works are used only for the two use cases above, i.e. at training as an additional objective, and at inference for speculative decoding. Our method on the other hand aims to additionally leverage MTP at inference to condition the next-token generation on the future information predicted by the MTP mechanism.

Planning & Lookahead Similar to MTP, other works also use future prediction as an additional training signal to improve planning capabilities of LLMs, but drop these mechanisms at inference (Zuhri et al., 2025; Mahajan et al., 2025). Yin et al. (2024) meanwhile include an auxiliary future prediction task, however they retain special planning tokens during inference to help guide the token generation based on a latent plan of the future. Wang et al. (2023) and Thankaraj et al. (2025) similarly make use of special planning/lookahead tokens to guide the generation process. In contrast to these methods which also help mitigate myopic generation, we do not use a high level plan of the future to guide the overall generation. Rather, we do a token-level lookahead and refinement in every forward pass, by projecting the immediate future using MTP.

Works such as Lu et al. (2022) and Nakshatri et al. (2025) propose lookahead mechanisms for constrained decoding, by rolling out future predictions across multiple beams (relying on discrete token sampling) to better inform the current token prediction. Our lookahead meanwhile occurs within a single forward pass and in latent space with minimal overhead (although it does not directly enable constraint satisfaction).

Latent & Implicit Thinking Most latent thinking approaches can be broadly categorized into token-wise and depth-wise approaches (Chen et al., 2025). Token-wise approaches such as Hao et al. (2024); Zhang et al. (2025); Wang et al. (2026); Kang et al. (2025) construct sequential chain-of-thought style reasoning traces in latent space. Our per-token thinking mechanism is orthogonal and operates within a single forward pass, adding no extra sequential steps or reasoning traces. More similar to our approach, Goyal et al. (2023); Herel & Mikolov (2024) allow extra computation budget per-token, however they use 'thinking' or 'pause' tokens which generally do not have a semantic target and only serve to provide additional computational pathways. Meanwhile, our additional tokens are grounded by their MTP objective. In contrast, depth-wise approaches typically apply transformer layers recursively to improve latent representations before decoding them to discrete tokens, allowing for dynamic compute allocation (Koishekenov et al., 2025; Geiping et al.,

2025). Such approaches are orthogonal to our work, and offer a natural intriguing future direction to combine with our method to allow for more adaptive refinement.

Our high level idea aligns closely with that of QuietSTaR (Zelikman et al., 2024). Whilst they condition the token generation on a combination of past context and an additional generated 'rationale', in our case, this 'rationale' is effectively constrained to being the latent future information predicted by the MTP mechanism. One of the observations by Zelikman et al. (2024) was that in some cases, the most useful rationales were near-continuations of the text, which strengthens the motivation behind our work that knowing what is coming in the future might better inform the current prediction. Compared to Quiet-STaR, our method requires no additional forward passes to generate thoughts, our 'thoughts' remain in latent space, and the actual method for inducing these 'thoughts' is very different (we use MTP to ground thoughts as being latent future information, whilst Quiet-STaR uses a reinforcement learning approach to learn useful thoughts freely).

3 METHOD

3.1 PRELIMINARIES

3.1.1 STANDARD AUTOREGRESSIVE MODELING

Language models aim to parameterize a distribution over a sequence of discrete tokens from a given vocabulary $\mathcal{V}$. Under the standard autoregressive paradigm, the joint distribution is factorized into the product of conditional next-token probabilities:

$$P_\theta(\mathbf{x}) = \prod_{t=0}^{T} P_\theta(x_{t+1} \mid x_{\leq t}) \,, \tag{1}$$

where $\mathbf{x} = (x_1, \ldots, x_T)$ is a token sequence with $x_t \in \mathcal{V}$. The model is trained to minimize the negative log-likelihood (NTP loss) over a dataset $\mathcal{D}$:

$$\mathcal{L}_{\text{NTP}}(\theta) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}} \left[-\sum_{t=0}^{T} \log P_\theta(x_{t+1} \mid x_{\leq t}) \right] . \tag{2}$$

During inference, sequences are generated by iteratively sampling the next token $\hat{x}_{t+1}$ conditioned strictly on the historical context $\hat{x}_{\leq t}$:

$$\hat{x}_{t+1} \sim P_\theta(\cdot \mid \hat{x}_{\leq t}) \,. \tag{3}$$

3.1.2 MULTI-TOKEN PREDICTION (MTP)

To augment the myopic NTP objective from Equation (2), numerous works inject an auxiliary MTP loss during training. Under MTP, the model is additionally trained to predict $P_\theta(x_{t+n} \mid x_{\leq t})$ for $n \in \{2, ..., k\}$, where k is the maximum prediction horizon. Thus, the overall training objective becomes a weighted sum of the primary NTP loss and the auxiliary MTP losses:

$$\mathcal{L}_{\text{MTP}}(\theta) = \lambda_1 \mathcal{L}_{\text{NTP}}(\theta) + \sum_{n=2}^{k} \lambda_n \mathcal{L}^{(n)}(\theta) \,, \tag{4}$$

where $\mathcal{L}^{(n)}(\theta)$ is the negative log-likelihood loss over $\mathcal{D}$ when predicting the n^{th} future token.

Whilst this general formulation holds for most MTP methods, the specific mechanism for predicting the future tokens can differ. For instance, multiple works attach additional prediction heads to a shared LLM backbone, where each head is tasked with predicting a future token (Gloeckle et al., 2024; Liu et al., 2024; 2025b). Gerontopoulos et al. (2025) on the other hand propose MuToR (Multi-Token Prediction with Registers), an alternative approach that involves inserting special 'register' tokens into the training sequence, which are given a future token prediction objective.

Crucially however, whilst these existing works leverage MTP during training to learn more future aware parameters θ, at inference, the MTP mechanisms are either dropped completely or are functionally isolated from the regular next-token generation process (if used for self-speculative decoding for example). Hence, the final generation mechanism reverts to Equation (3), whereby the next-token is conditioned only on the past context and is not explicitly aware of any future information beyond what was implicitly learned by θ.

3.2 OUR APPROACH

We aim to address this inference gap by additionally conditioning the next-token generation on the latent future information $\Phi_{\text{future}}(\hat{x}_{\leq t})$ predicted by the MTP mechanism:

$$\hat{x}_{t+1} \sim P_\theta \left(\cdot \mid \hat{x}_{\leq t},\ \Phi_{\text{future}}(\hat{x}_{\leq t}) \right) \ . \tag{5}$$

Thus, we do not inherently break the causal structure of the model, since the future information itself is conditioned purely on historical context $x_{\leq t}$. We simply propose to additionally leverage this future information during inference to allow the current token to be explicitly aware of what is projected to appear ahead, so that it can make a more informed prediction for the next token.

Our method, LaLoR, builds upon MuToR (Gerontopoulos et al., 2025) for a few key reasons:

1. Using special 'register' tokens for future token prediction means that we have access to intermediate latent future states, which is less trivial to obtain with other methods that only predict future tokens at the final prediction head.
2. This allows us to easily introduce a lookahead mechanism by modifying the attention mask, enabling future projection and lookahead refinement *within a single forward pass*.
3. There is virtually no parameter overhead or core architectural change compared to other MTP methods. We simply modify attention masks and add one additional embedding vector for the register token, independent of the max horizon k. Meanwhile, alternative MTP methods which use additional heads require a large parameter overhead that typically scales with k.
4. Given that inference is typically memory-bounded (Yuan et al., 2024), using additional register tokens for future projection at inference time incurs minimals latency overhead. Additionally, we maintain full compatibility with recent methods for highly parallelized drafting and verification, such as those proposed by Liu et al. (2025a) and Samragh et al. (2025), opening the potential for significant inference speedups as well.

3.2.1 MTP SETUP

Let $\mathbf{x} = (x_1, \ldots, x_T)$ be a given token sequence of length T (where we refer to each x_t as a 'regular' token). We introduce special register tokens $r_{t,d}$ which are given a future token prediction task (as opposed to the next-token prediction target of regular tokens). Whilst all register tokens share the same input embedding as in Gerontopoulos et al. (2025), each register $r_{t,d}$ is uniquely determined by an offset $d \in \{2, \ldots, d_{\text{max}}\}$, where d_{max} is the maximum prediction horizon (equivalent to k from earlier), and an index $t \in \{1, \ldots, T\}$, to be interpreted as the position of the 'owner' regular token x_t (i.e., the closest preceding regular token). Thus, the prediction target of a register token $r_{t,d}$ becomes $x_{(t+d)}$, and registers are only valid if $t + d \leq T$. This setup only introduces D new parameters to the model, where D is the size of a token embedding vector.

Register Token Insertion During training, in order to keep a fixed token budget when comparing to baseline methods, we introduce a register budget $B_r \in [0, 1]$. For each training sequence $\mathbf{x}$, we then generate an augmented sequence $\mathbf{x}'$ by first truncating the final $\lfloor B_r T \rfloor$ tokens from $\mathbf{x}$, and then randomly inserting $\lfloor B_r T \rfloor$ register tokens. Since each register token can be uniquely characterized by a tuple (t, d), we sample $\lfloor B_r T \rfloor$ such tuples uniformly at random without replacement from the set $\{(\tau, \delta) \in \{1, ..., T\} \times \{2, ..., d_{\text{max}}\} : \tau + \delta \leq T\}$, and then insert a register $r_{t,d}$ immediately after x_t for each tuple (t, d). Thus, each regular token may have between 0 and $d_{\text{max}} - 1$ register tokens appended to it, each predicting the d^{th}-next token. For example, with $d_{\text{max}} = 4$, a possible final sequence could look like

$$\mathbf{x}' = (x_1, \underline{r_{1,2}}, \underline{r_{1,4}}, x_2, x_3, \underline{r_{3,3}}, x_4, \underline{r_{4,2}}, x_5, \underline{r_{5,2}}, \underline{r_{5,3}}, \underline{r_{5,4}}, \ldots, x_{\lceil T(1-B_r) \rceil}) \ ,$$

with the registers becoming more sparse with a lower budget B_r.

Position IDs We maintain each regular token x_t to have a 0-indexed position ID of $t - 1$, whilst each register $r_{t,d}$ has position ID of $t + d - 2$. As such, registers are given the same position ID as the regular token which has the same next-token prediction target $x_{(t+d)}$.

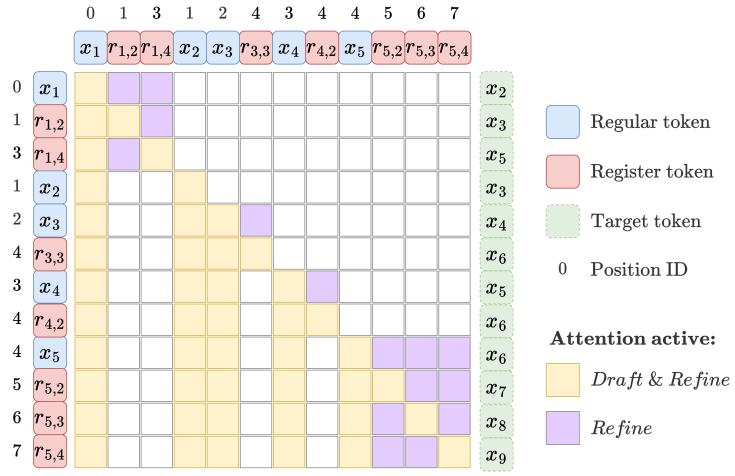


Figure 1: Example of an attention mask constructed for a sequence with randomly inserted register tokens. The mask in yellow is a standard causal mask modified for the inclusion of register tokens, which effectively can only be attended to by themselves. This is mask we use in the *draft* stage. The purple mask shows the new attention connections that are enabled during the *refine* stage. Specifically, there is now fully bidirectional attention between each regular token and its corresponding register tokens.

Attention Mask For our base MTP setup, we follow the same rules for creating the attention mask as is done in MuToR (Gerontopoulos et al., 2025): 1) regular tokens do not attend to any register tokens, and 2) register tokens cannot attend to any other register tokens (except themselves). Otherwise, we maintain a standard causal mask. An example attention mask for the augmented sequence $\mathbf{x}'$ above is shown in Figure 1 (in yellow), along with the position IDs and prediction targets of each token.

Training Objective Our MTP training objective follows the same structure as the general formulation in Equation (4). Then, $\mathcal{L}_{\text{NTP}}(\theta)$ is the standard negative log-likelihood loss from Equation (2) calculated across all regular tokens. When constructing the auxiliary loss, we weight registers at all offsets equally. Thus given an augmented dataset $\hat{\mathcal{D}}'$ (with registers inserted), we have:

$$\mathcal{L}_{\text{Reg}}(\theta) = \mathbb{E}_{\mathbf{x}' \sim \hat{\mathcal{D}}'} \left[-\sum_{t=0}^{T} \sum_{d=2}^{d_{\max}} \mathbf{1}_{\{\mathrm{r}_{\mathrm{t,d}} \in \mathbf{x}'\}} \log P_\theta(x_{t+d} \mid x_{\leq t}, r_{t,d}) \right] . \tag{6}$$

The final loss is balanced between $\mathcal{L}_{\text{NTP}}(\theta)$ and $\mathcal{L}_{\text{Reg}}(\theta)$ using a factor $\alpha \in (0, 1)$:

$$\mathcal{L}_{\text{MTP}} = \underbrace{(1-\alpha)\mathcal{L}_{\text{NTP}}(\theta)}_{\text{Regular Tokens}} + \underbrace{\alpha \mathcal{L}_{\text{Reg}}(\theta)}_{\text{Register Tokens}} \tag{7}$$

3.2.2 Latent Lookahead Refinement

The key novelty of our method is to introduce a mechanism for latent lookahead refinement on top of the base MTP setup. At a high level, the idea is to use the first part of the network to *draft* an early representation for the next- and future-tokens (in parallel), whilst the final part of the network allows the current token to look ahead at the future projections to *refine* its next-token prediction accordingly.

More concretely, let L be the total number of layers in our transformer model, and ℓ_r denote the number of refinement layers. The first $(L - \ell_r)$ layers of the model are responsible for the *draft* stage, using the (modified) causal attention mask outlined in Section 3.2.1. However, for the final ℓ_r layers (the *refine* stage), we modify this attention mask to allow fully bidirectional attention between each regular token and its corresponding register tokens. This is depicted pictorially in

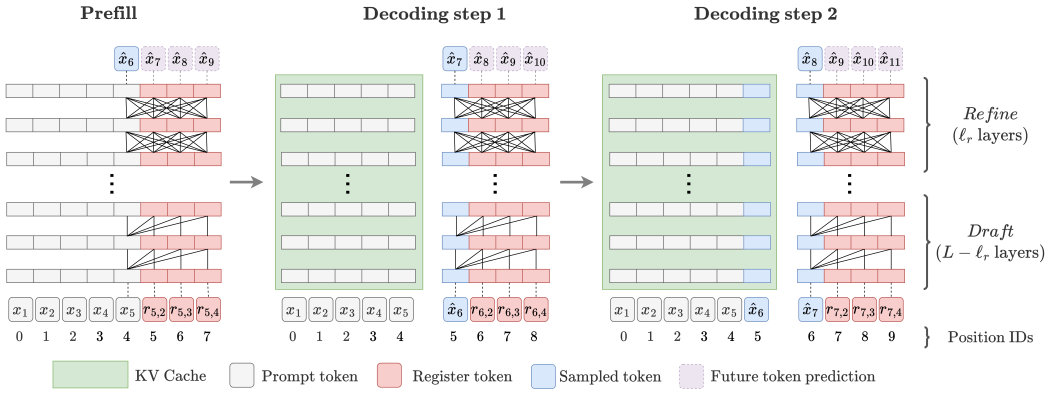


Figure 2: Inference with LaLoR using $d_{\text{max}} = 4$. At every step (including the first prefill step), we append a set of 3 registers (predicting the 2^{nd}-, 3^{rd}- and 4^{th}-next tokens) to the last prompt/sampled token. For the *draft* stage, we use the standard MuToR attention masking scheme, whereby register tokens can attend to previous regular tokens (and themselves), whilst regular tokens cannot attend to register tokens. In the *refine* stage, we introduce fully bidirectional attention between the final prompt/sampled token and the set of register tokens that have been appended to it. Each sampled token is thus conditioned not only on past context, but also on latent future information encoded by the register tokens. This inference scheme allows for exact KV caching, as well as potential for speculative decoding schemes by utilising the future token predictions. Note: wherever no attention pattern is shown, we assume standard causal attention.

Figure 1. Thus, since we simply modify the attention mask between our two model stages, our method enables a form of per-token latent thinking that occurs within a single forward pass and with minimal computational or memory overhead (see Appendix E for ablations on inference latency).

Inference Once we train the network with the register tokens and the *draft* & *refine* stages, during generation, we can simply append a set of $d_{\text{max}} - 1$ registers with offsets $d \in \{2, ..., d_{\text{max}}\}$ to the last token of the input at each step. These registers will project future information in the *draft* stage, and in the *refine* stage, the current token is explicitly enabled to see this information. Thus, the next-token prediction for x_t is no longer only conditioned on previous context $x_{<t}$, but also on latent future information $\Phi_{\text{future}}(x_{<t})$, where

$$\Phi_{\text{future}}(x_{<t}) = g^{(L-\ell_r+1):L}_{(t+1):(t+d_{\text{max}}-1)} \ , \tag{8}$$

where $g^{(L-\ell_r+1):L}_{(t+1):(t+d_{\text{max}}-1)}$ represents the hidden states of the register tokens $\{r_{t,2}, ...r_{t,d_{max}}\}$ across the final ℓ_r layers $\{L - \ell_r + 1, ..., L\}$ of the model. This allows us to achieve the future-informed sampling from Equation (5). An illustration of the full inference procedure is shown in more detail in Figure 2.

Note that in principle, this future-informed inference does not actually improve upon an optimal next-token predictor, given that the latent future information itself is conditioned only on past context. Instead, our method is effectively adding a structural inductive bias to the model through the *draft* and *refine* stages.

4 Experiments

4.1 Setup

In this paper, we focus on a toy continual pretraining setting (from a base model trained with a standard NTP objective), where we aim to improve the math capability of the model and test whether LaLoR can offer improvements over standard NTP/MTP training and inference. We choose to focus on math since this is a domain that could benefit from a lookahead mechanism (e.g. to solve issues such as cascading carries (Baeumel et al., 2025)).

Models We use the pretrained SmolLM2 base models (Allal et al., 2025), which are a family of open small language models that are competitively strong for their size (including in math capability). We mostly focus our experiments on the 1.7B parameter model, but also conduct some experimentation on the 360M variant.

Data We use a high-quality data mix comparable to what was used in the final stages of pretraining SmolLM2-1.7B (Allal et al., 2025), but we upsample the math data to try improve math adaptation. We use a mix of 40% high-quality math data using the Nemotron-CC-Math dataset (Rabeeh Karimi Mahabadi, 2025; NVIDIA et al., 2025), and 60% high-quality English web data from DCLM-Edu (Allal et al., 2025) and FineWeb-Edu (Lozhkov et al., 2024). More details about our data mix can be found in Appendix A. Most of our experiments (unless otherwise specified) are conducted using a 2048 context length with sequence packing (we ensure that there is no cross-document attention, and that register targets remain within the same document).

Training Unless otherwise specified, we train SmolLM2 1.7B on 10B tokens. We use a Warmup-Stable-Decay (WSD) learning rate scheduler with a peak learning rate of 2.5×10^{-4}, with 750 warmup steps and the final 10% of training reserved for cooldown. For all experiments with register tokens (MTP baselines and LaLoR), we use $d_{\text{max}} = 4$, since both Gloeckle et al. (2024) and Gerontopoulos et al. (2025) found a max horizon of 4 to be effective for MTP (although we aim to study the effect of this parameter in future work). Unless otherwise specified, we also use a register budget B_r of 0.1, and loss factor α of 0.1 (as explained in Appendix C). For LaLoR, we reserve the final 25% of layers for the *refine* stage ($\ell_r = 6$ out of $L = 24$ layers for the 1.7B model). This was chosen to allow the internal representations to converge enough to be meaningful, whilst also leaving enough layers for refinement to have an influence, although this is likewise a key hyperparameter we aim to study in future work. Further details about our training setup can be found in Appendix B.

Evaluation We use LM Eval Harness (Gao et al., 2024) to evaluate our models on a set of math and reasoning & commonsense benchmarks, to measure if the model improved on math whilst maintaining its general capabilities. For math, we use GSM8K (Cobbe et al., 2021), GSM-Plus (Li et al., 2024), and MATH500 (Hendrycks et al., 2021) with few-shot prompting. We refer to these as GSM8K, Plus and MATH respectively. For reasoning & commonsense, we use ARC (Clark et al., 2018) (both Easy and Challenge), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020) and CommonsenseQA (Talmor et al., 2019). We refer to these as ARC-E, ARC-C, Hella, PIQA, and CQA respectively. We report performances on individual tasks, macro-averages for each category of task, and an overall macro-average. Further detail on our evaluation configuration is provided in Appendix D.

4.2 Baseline Results

The results of our basic LaLoR setup compared to both NTP and MTP baselines is shown in Figure 3. As expected, all methods improve on math benchmarks, whilst mostly maintaining reasoning & commonsense capabilties. CommonsenseQA (CQA) seems to benefit from an MTP objective. Beyond this however, it is difficult to draw concrete conclusions from these results considering the performance differences between methods are relatively small, and how much the evaluations fluctuate across checkpoints. For example, we see a slight improvement in the total macro-average performance for the MTP baseline after the final cooldown, largely due to a performance spike on MATH; however, the MTP baseline has very high variance on MATH across checkpoints suggesting this difference may be dominated by noise.

Regardless, we see that LaLoR does not seem to improve performance over standard MTP, following a similar trajectory on average. Additionally, disabling the refinement at inference time does not affect performance significantly. To investigate further, we analyse the lookahead attention scores and find that during training, the lookahead attention is attenuated, as shown in Figure 7 of Appendix F. This result alone however is difficult to interpret, because it could indicate that the model is learning to ignore the registers, or the lower attention scores might be learned based on the utility of the registers.

Coupled with the observation that refinement does not seem to impact downstream performance, we have two main hypotheses. The first hypothesis is that since the model was pretrained with a

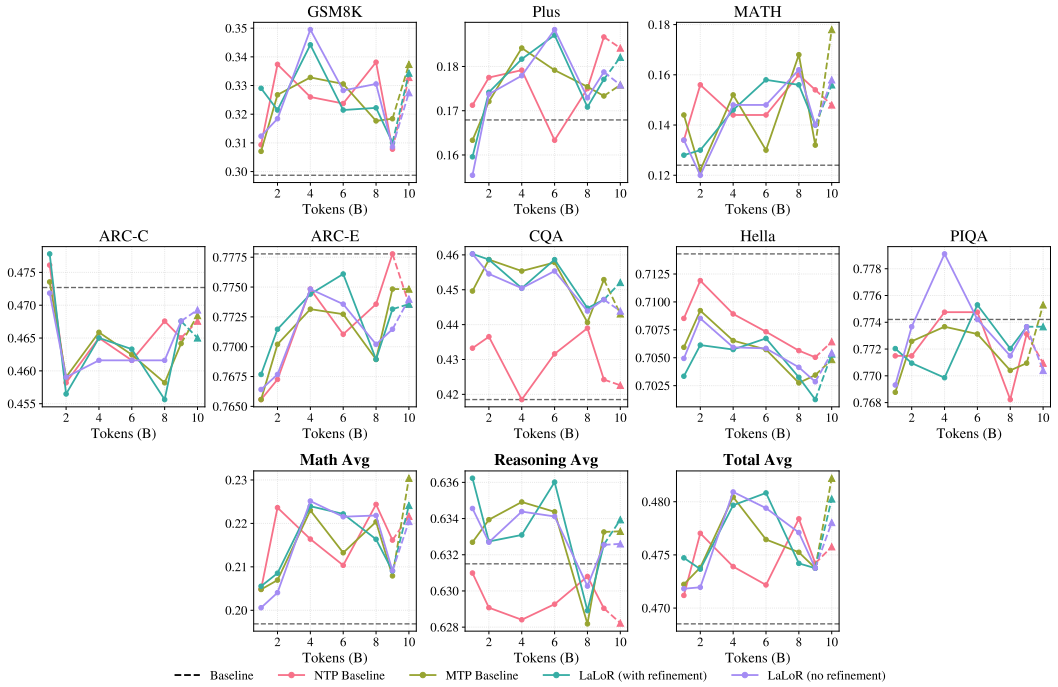


Figure 3: Evaluation results for SmolLM2 1.7B throughout the 10B token training. The NTP baseline is trained using a vanilla NTP objective, whilst the MTP baseline is trained with the additional register loss but without the lookahead mechanism. Both LaLoR results are trained with MTP and lookahead, but inference is done either with or without refinement enabled. We perform evaluations at multiple checkpoints during the stable phase (circles), whilst the final result (triangle) is measured after the learning rate cooldown. All methods improve over the base model in math benchmarks, and introducing an MTP objective seems to help with CommonsenseQA. However, there appears to be minimal difference on other tasks between methods, and the results are quite noisy across training.

significant amount of data (11T tokens) in a purely causal fashion, it is perhaps difficult to induce this new lookahead behaviour when the model can already predict well based on prior context alone, and it simply learns to ignore this new signal instead during our short 10B token training. This would not be surprising considering that related work Quiet-STaR initially faced similar issues with the model learning to ignore the generated thoughts (Zelikman et al., 2024) in their continual pretraining setup. The second hypothesis is that the lookahead attention is being learned, however its utility is limited and it does not actually help with improving the current token's prediction in a meaningful way. Below, we outline additional experiments to test these hypotheses.

4.3 Attention Bias to Force Lookahead

We try introducing a bias in the attention mask to explicitly encourage the model to look ahead, to test if it is indeed simply learning to ignore the registers because it has no incentive to look ahead. We introduce this bias with a scheduler, whereby we begin with a strong negative bias (effectively disabling the lookahead when the latent future information is not learnt yet and thus is likely to just be noise), and then gradually introduce positive bias to encourage the model to look ahead as the representation improves. We eventually cool down the bias to zero to allow the model to learn more freely, after having encouraged the model to learn that there is useful information to attend to in the future. Further details about this experiment are provided in Appendix G. Analysis of the lookahead attention behavior throughout these different stages of training, however, indicates that the model does indeed seem to be learning to attend to the register tokens, as it tries to maintain relatively stable attention scores in spite of the varying biases throughout training, as described in Figure 8. Thus, it seems that our first hypothesis does not hold, and it is more likely that the lack of performance gains indicates that the latent future information simply has low utility. Additionally, since the model

seems to learn the attentions naturally, and introducing the bias simply causes the model to learn to compensate for this bias, we do not apply biases to the attention mask in any further experiments.

4.4 Addressing Register Token Sparsity

One potential reason for the lookahead having low utility at inference could be if the model has not learned to effectively leverage it for refinement, given the sparsity of register tokens during training. In our experiments, we use a register budget of only 10% ($B_r = 0.1$), so the majority of regular tokens do not have corresponding register tokens, and thus there is a very sparse signal to learn effective refinement behavior. Additionally, since we use a randomized register insertion strategy, even fewer regular tokens will have a dense signal of multiple register tokens to attend to. Thus, there is also an additional train-inference discrepancy when using refinement, since during inference we always append a complete set of registers, which the model may not have learned to leverage effectively. As such, we conduct additional experiments where we adjust the register insertion strategy to insert complete sets of registers with all offsets $d \in \{2, ..., d_{\text{max}}\}$ for a random subset of regular tokens, as well as not constraining the total token budget and instead add T registers to sequences of length T, effectively doubling the training sequence length. We find that neither of these modifications improve LaLoR compared to the MTP baseline, as shown in Figure 9, indicating that register sparsity, whilst possibly still relevant, is likely not the key reason for the low utility of the lookahead. Further details of this experiment are in provided in Appendix H.

4.5 Intermediate Loss to Improve Latent Representations

Another possible reason for the low utility of the lookahead mechanism during inference could be if the latent future representations themselves are not helpful to the current token's prediction. If these representations are poor, we would expect the model to have lower attention to them, and for there to be less useful information to extract, diminishing the benefits of refinement. Hence, in order to try improve the intermediate representations, we experiment with introducing an additional loss term. Specifically, we aim to maximize the cosine similarity between the hidden state of each register token $r_{t,d}$ and the hidden state of the regular token it is imitating x_{t+d-1}, which we take to be the ideal representation for the registers to learn. This loss is described in more detail in Appendix I.1.

Although Figure 10 shows no meaningful differences in downstream performance when this intermediate loss is introduced, our analysis in Appendix I reveals a few key insights. Firstly, this intermediate loss can marginally improve the representations at certain (but not all) layers (as shown in Figure 11). Second, these improvements in representation correlate with higher attention scores (Figure 12). Thus, we have another strong indication that the model is indeed learning to attend to register tokens based on the quality of the representations. Although this does not translate into downstream improvements in the current setup, it seems likely that the potential utility of the lookahead refinement is strongly tied to the strength of the latent future representations.

4.6 Performance by Model Scale

Prior works on MTP have found that the benefits of MTP scale with model size, and MTP can in fact hurt the performance of smaller models (Gloeckle et al., 2024; Nagarajan et al., 2025), likely due to a limited capacity to learn rich representations. Since our method is heavily reliant on the quality of the MTP, we hypothesize that the small models we work with do not have the capacity to learn strong future representations, meaning the intermediate latent future information may not be useful for lookahead refinement. To test whether MTP capability varies with scale under our scheme, we run experiments to compare NTP against MTP and LaLoR on the smaller 360M parameter model of the SmolLM2 family. Results in Figure 13, whilst not entirely conclusive, do seem to indicate marginally worse performance when introducing the MTP loss, including on CommonsenseQA (CQA) which MTP benefited for the 1.7B model. Analysis of the top-k prediction accuracies of these two model scales in Figure 14 further suggests that MTP capability will likely improve with model scale. Thus, it is possible that scaling to larger models might yield more meaningful latent future representations that can better inform lookahead refinement.

5 CONCLUSION

In this work, we propose LaLoR, a novel latent thinking method that builds upon recent token-based MTP methods to introduce Latent Lookahead Refinement. Specifically, we describe a mechanism for future-informed generation of the next-token, which occurs within a single forward pass and incurs minimal inference overhead. Although early experiments with our method do not yield downstream performance benefits over standard MTP, based on our analyses, we hypothesize that this can likely be attributed to weak latent future representations of the models we train, which would be uninformative for lookahead refinement, and we believe scaling up could be beneficial. Overall, we hope our hypotheses and analyses can provide key insights to the scientific community and allow for future work to improve upon the proposed method.

Limitations Given that our experiments do not demonstrate clear performance benefits of LaLoR over MTP, the method is largely unverified. Thus, its practicality remains limited until we can produce clear evidence that it is effective. Additionally, although we hypothesize that scaling up the models could help, there is no guarantee of this, and running experiments at larger scale is expensive. Furthermore, in our current experimental setup, the downstream evaluations are noisy, making it difficult to conclude if the relatively small performance differences we see are due to a specific method or simply dominated by noise.

In terms of the method itself, whilst LaLoR is, in principle, very flexible to the training set-up (e.g. training from scratch vs continual pretraining, different base models etc.), there exists some limitations. For example, the reliance on custom attention masks makes this approach inherently incompatible with state-space models, or even more recent hybrid models (e.g. Nemotron-3 (NVIDIA, 2025)) which mix linear attention layers with full attention. Also, this means we cannot use optimized attention kernels such as FlashAttention (Dao et al., 2022) out of the box. Although we did not do so in this work, it should however be possible to leverage FlexAttention (Dong et al., 2024) for faster attention calculation, given the structured masks we use.

Finally, in order to have access to intermediate future hidden states, our method relies on token-based MTP. Whilst it is a promising recent approach, using additional heads seems to be the prevailing MTP approach in recent major model releases (Liu et al., 2024; NVIDIA, 2025; Xiao et al., 2026; Zeng et al., 2026), meaning our method is not a direct extension of the popular MTP paradigm.

Future Work Our main direction for future work will be trying to improve the latent future representations, such that the future signals we inject into the inference procedure might actually help refine the predictions. As discussed previously, the most promising direction for this seems to be scaling up the model size. Of course, scaling up the model size also scales up costs, thus we will also explore additional strategies for improving the latent future representations at this small model scale. For example, the decision to not allow attention between register tokens (with the same owner token) in the *draft* stage is somewhat arbitrary – in our setup, there is no reason we cannot allow bidirectional attention from the onset between register tokens with the same owner, since they are conditioned on the same shared history, and could potentially benefit from the shared attention throughout.

Furthermore, given that we could not clearly distinguish any potential performance gains due to noisy evaluations, changing the experimental setup could potentially help. For example, performing continual pretraining on an earlier checkpoint where benchmark results are still improving during pretraining could potentially reveal differences more clearly, particularly considering we only trained for 10B tokens on a model which was pretrained for 11T tokens. Also, benchmarking on some more specialized synthetic tasks could allow more targeted evaluations on the potential efficacy of the lookahead mechanism, since it is possible it does not impact every task equally.

There are also a number of key parameters we have not yet studied in this work which we wish to explore. For example, once we have evidence that the method can work, it would be interesting to investigate how far we can realistically scale up the future prediction horizon, as well as understand the effect of varying the number of refinement layers in detail. Additionally, although we claim the method maintains compatibility with highly parallel drafting and verification schemes such as that proposed by Liu et al. (2025a), concretely implementing this to measure how much speedup we can achieve in practice would strengthen this claim.

REFERENCES

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, et al. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.

Gregor Bachmann and Vaishnavh Nagarajan. The pitfalls of next-token prediction. *arXiv preprint arXiv:2403.06963*, 2024.

Tanja Baeumel, Josef van Genabith, and Simon Ostermann. The lookahead limitation: Why multi-operand addition is hard for llms. *arXiv preprint arXiv:2502.19981*, 2025.

Elie Bakouch, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Lewis Tunstall, Carlos Miguel Patiño, Edward Beeching, Aymeric Roucher, Aksel Joonas Reedi, Quentin Gallouédec, Kashif Rasul, Nathan Habib, Clémentine Fourrier, Hynek Kydlicek, Guilherme Penedo, Hugo Larcher, Mathieu Morlon, Vaibhav Srivastav, Joshua Lochner, Xuan-Son Nguyen, Colin Raffel, Leandro von Werra, and Thomas Wolf. SmolLM3: smol, multilingual, long-context reasoner. `https://huggingface.co/blog/smollm3`, 2025.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D Lee, Deming Chen, and Tri Dao. Medusa: Simple llm inference acceleration framework with multiple decoding heads. *arXiv preprint arXiv:2401.10774*, 2024.

Hao Mark Chen, Wayne Luk, Ka Fai Cedric Yiu, Rui Li, Konstantin Mishchenko, Stylianos I Venieris, and Hongxiang Fan. Hardware-aware parallel prompt decoding for memory-efficient acceleration of llm inference. *arXiv preprint arXiv:2405.18628*, 2024.

Xinghao Chen, Anhao Zhao, Heming Xia, Xuan Lu, Hanlin Wang, Yanjun Chen, Wei Zhang, Jian Wang, Wenjie Li, and Xiaoyu Shen. Reasoning beyond language: A comprehensive survey on latent chain-of-thought reasoning. *arXiv preprint arXiv:2505.16782*, 2025.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457, 2018.

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

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

Juechu Dong, Boyuan Feng, Driss Guessous, Yanbo Liang, and Horace He. Flex attention: A programming model for generating optimized attention kernels. *arXiv preprint arXiv:2412.05496*, 2024.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL `https://zenodo.org/records/12608602`.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach, 2025. URL `https://arxiv.org/abs/2502.05171`.

Anastasios Gerontopoulos, Spyros Gidaris, and Nikos Komodakis. Multi-token prediction needs registers. *arXiv preprint arXiv:2505.10518*, 2025.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. *arXiv preprint arXiv:2404.19737*, 2024.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023.

Alex Hägele, Elie Bakouch, Atli Kosson, Leandro Von Werra, Martin Jaggi, et al. Scaling laws and compute-optimal training beyond fixed training durations. *Advances in Neural Information Processing Systems*, 37:76232–76264, 2024.

Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training large language models to reason in a continuous latent space. *arXiv preprint arXiv:2412.06769*, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *Advances in neural information processing systems*, 2021.

David Herel and Tomas Mikolov. Thinking tokens for language modeling, 2024. URL https://arxiv.org/abs/2405.08644.

Adam Ibrahim, Benjamin Thérien, Kshitij Gupta, Mats L Richter, Quentin Anthony, Timothée Lesort, Eugene Belilovsky, and Irina Rish. Simple and scalable strategies to continually pre-train large language models. *arXiv preprint arXiv:2403.08763*, 2024.

Haoqiang Kang, Yizhe Zhang, Nikki Lijing Kuang, Nicklas Majamaki, Navdeep Jaitly, Yi-An Ma, and Lianhui Qin. Ladir: Latent diffusion enhances llms for text reasoning, 2025. URL https://arxiv.org/abs/2510.04573.

Yeskendir Koishekenov, Aldo Lipani, and Nicola Cancedda. Encode, think, decode: Scaling test-time reasoning with recursive latent thoughts, 2025. URL https://arxiv.org/abs/2510.07358.

Hynek Kydlicek, Alina Lozovskaya, Nathan Habib, and Clémentine Fourrier. Fixing open llm leaderboard with math-verify, 2025.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. Gsm-plus: A comprehensive benchmark for evaluating the robustness of llms as mathematical problem solvers, 2024. URL https://arxiv.org/abs/2402.19255.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Jan Kautz, Ce Zhang, and Pavlo Molchanov. Tidar: Think in diffusion, talk in autoregression. *arXiv preprint arXiv:2511.08923*, 2025a.

Xiaohao Liu, Xiaobo Xia, Weixiang Zhao, Manyi Zhang, Xianzhi Yu, Xiu Su, Shuo Yang, See-Kiong Ng, and Tat-Seng Chua. L-mtp: Leap multi-token prediction beyond adjacent context for large language models. *arXiv preprint arXiv:2505.17505*, 2025b.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Ximing Lu, Sean Welleck, Peter West, Liwei Jiang, Jungo Kasai, Daniel Khashabi, Ronan Le Bras, Lianhui Qin, Youngjae Yu, Rowan Zellers, et al. Neurologic a* esque decoding: Constrained text generation with lookahead heuristics. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 780–799, 2022.

Divyat Mahajan, Sachin Goyal, Badr Youbi Idrissi, Mohammad Pezeshki, Ioannis Mitliagkas, David Lopez-Paz, and Kartik Ahuja. Beyond multi-token prediction: Pretraining llms with future summaries. *arXiv preprint arXiv:2510.14751*, 2025.

Vaishnavh Nagarajan, Chen Henry Wu, Charles Ding, and Aditi Raghunathan. Roll the dice & look before you leap: Going beyond the creative limits of next-token prediction. *arXiv preprint arXiv:2504.15266*, 2025.

Nishanth Sridhar Nakshatri, Shamik Roy, Rajarshi Das, Suthee Chaidaroon, Leonid Boytsov, and Rashmi Gangadharaiah. Constrained decoding with speculative lookaheads. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4681–4700, 2025.

NVIDIA. Nvidia nemotron 3: Efficient and open intelligence, 2025. URL https://arxiv.org/abs/2512.20856. White Paper.

NVIDIA, :, Aarti Basant, Abhijit Khairnar, Abhijit Paithankar, Abhinav Khattar, Adithya Renduchintala, Aditya Malte, Akhiad Bercovich, Akshay Hazare, Alejandra Rico, Aleksander Ficek, Alex Kondratenko, Alex Shaposhnikov, Alexander Bukharin, Ali Taghibakhshi, Amelia Barton, Ameya Sunil Mahabaleshwarkar, Amy Shen, Andrew Tao, Ann Guan, Anna Shors, Anubhav Mandarwal, Arham Mehta, Arun Venkatesan, Ashton Sharabiani, Ashwath Aithal, Ashwin Poojary, Ayush Dattagupta, Balaram Buddharaju, Banghua Zhu, Barnaby Simkin, Bilal Kartal, Bita Darvish Rouhani, Bobby Chen, Boris Ginsburg, Brandon Norick, Brian Yu, Bryan Catanzaro, Charles Wang, Charlie Truong, Chetan Mungekar, Chintan Patel, Chris Alexiuk, Christian Munley, Christopher Parisien, Dan Su, Daniel Afrimi, Daniel Korzekwa, Daniel Rohrer, Daria Gitman, David Mosallanezhad, Deepak Narayanan, Dima Rekesh, Dina Yared, Dmytro Pykhtar, Dong Ahn, Duncan Riach, Eileen Long, Elliott Ning, Eric Chung, Erick Galinkin, Evelina Bakhturina, Gargi Prasad, Gerald Shen, Haifeng Qian, Haim Elisha, Harsh Sharma, Hayley Ross, Helen Ngo, Herman Sahota, Hexin Wang, Hoo Chang Shin, Hua Huang, Iain Cunningham, Igor Gitman, Ivan Moshkov, Jaehun Jung, Jan Kautz, Jane Polak Scowcroft, Jared Casper, Jian Zhang, Jiaqi Zeng, Jimmy Zhang, Jinze Xue, Jocelyn Huang, Joey Conway, John Kamalu, Jonathan Cohen, Joseph Jennings, Julien Veron Vialard, Junkeun Yi, Jupinder Parmar, Kari Briski, Katherine Cheung, Katherine Luna, Keith Wyss, Keshav Santhanam, Kezhi Kong, Krzysztof Pawelec, Kumar Anik, Kunlun Li, Kushan Ahmadian, Lawrence McAfee, Laya Sleiman, Leon Derczynski, Luis Vega, Maer Rodrigues de Melo, Makesh Narsimhan Sreedhar, Marcin Chochowski, Mark Cai, Markus Kliegl, Marta Stepniewska-Dziubinska, Matvei Novikov, Mehrzad Samadi, Meredith Price, Meriem Boubdir, Michael Boone, Michael Evans, Michal Bien, Michal Zawalski, Miguel Martinez, Mike Chrzanowski, Mohammad Shoeybi, Mostofa Patwary, Namit Dhameja, Nave Assaf, Negar Habibi, Nidhi Bhatia, Nikki Pope, Nima Tajbakhsh, Nirmal Kumar Juluru, Oleg Rybakov, Oleksii Hrinchuk, Oleksii Kuchaiev, Oluwatobi Olabiyi, Pablo Ribalta, Padmavathy Subramanian, Parth Chadha, Pavlo Molchanov, Peter Dykas, Peter Jin, Piotr Bialecki, Piotr Januszewski, Pradeep Thalasta, Prashant Gaikwad, Prasoon Varshney, Pritam Gundecha, Przemek Tredak, Rabeeh Karimi Mahabadi, Rajen Patel, Ran El-Yaniv, Ranjit Rajan, Ria Cheruvu, Rima Shahbazyan, Ritika Borkar, Ritu Gala, Roger Waleffe, Ruoxi Zhang, Russell J. Hewett, Ryan Prenger, Sahil Jain, Samuel Kriman, Sanjeev Satheesh, Saori Kaji, Sarah Yurick, Saurav Muralidharan, Sean Narenthiran, Seonmyeong Bak, Sepehr Sameni, Seungju Han, Shanmugam Ramasamy, Shaona Ghosh, Sharath Turuvekere Sreenivas, Shelby Thomas, Shizhe Diao, Shreya Gopal, Shrimai Prabhumoye, Shubham Toshniwal, Shuoyang Ding, Siddharth Singh, Siddhartha Jain, Somshubra Majumdar, Soumye Singhal, Stefania Alborghetti, Syeda Nahida Akter, Terry Kong, Tim Moon, Tomasz Hliwiak, Tomer Asida, Tony Wang, Tugrul Konuk, Twinkle Vashishth, Tyler Poon, Udi Karpas, Vahid Noroozi, Venkat Srinivasan, Vijay Korthikanti, Vikram

Fugro, Vineeth Kalluru, Vitaly Kurin, Vitaly Lavrukhin, Wasi Uddin Ahmad, Wei Du, Wonmin Byeon, Ximing Lu, Xin Dong, Yashaswi Karnati, Yejin Choi, Yian Zhang, Ying Lin, Yonggan Fu, Yoshi Suhara, Zhen Dong, Zhiyu Li, Zhongbo Zhu, and Zijia Chen. Nvidia nemotron nano 2: An accurate and efficient hybrid mamba-transformer reasoning model, 2025. URL https://arxiv.org/abs/2508.14444.

Shrimai Prabhumoye Mostofa Patwary Mohammad Shoeybi Bryan Catanzaro Rabeeh Karimi Mahabadi, Sanjeev Satheesh. Nemotron-cc-math: A 133 billion-token-scale high quality math pretraining dataset. 2025. URL https://arxiv.org/abs/2508.15096.

Mohammad Samragh, Arnav Kundu, David Harrison, Kumari Nishu, Devang Naik, Minsik Cho, and Mehrdad Farajtabar. Your llm knows the future: Uncovering its multi-token prediction potential. *arXiv preprint arXiv:2507.11851*, 2025.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1421. URL https://aclanthology.org/N19-1421.

Abitha Thankaraj, Yiding Jiang, J Zico Kolter, and Yonatan Bisk. Looking beyond the next token. *arXiv preprint arXiv:2504.11336*, 2025.

Jiecong Wang, Hao Peng, and Chunyang Liu. Latent chain-of-thought as planning: Decoupling reasoning from verbalization, 2026. URL https://arxiv.org/abs/2601.21358.

Xinyi Wang, Lucas Caccia, Oleksiy Ostapenko, Xingdi Yuan, William Yang Wang, and Alessandro Sordoni. Guiding language model reasoning with planning tokens. *arXiv preprint arXiv:2310.05707*, 2023.

Bangjun Xiao, Bingquan Xia, Bo Yang, Bofei Gao, Bowen Shen, Chen Zhang, Chenhong He, Chiheng Lou, Fuli Luo, Gang Wang, et al. Mimo-v2-flash technical report. *arXiv preprint arXiv:2601.02780*, 2026.

Yongjing Yin, Junran Ding, Kai Song, and Yue Zhang. Semformer: Transformer language models with semantic planning. *arXiv preprint arXiv:2409.11143*, 2024.

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, et al. Llm inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*, 2024.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chengxing Xie, Cunxiang Wang, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Zhen Zhang, Xuehai He, Weixiang Yan, Ao Shen, Chenyang Zhao, Shuohang Wang, Yelong Shen, and Xin Eric Wang. Soft thinking: Unlocking the reasoning potential of llms in continuous concept space, 2025. URL https://arxiv.org/abs/2505.15778.

Zayd MK Zuhri, Erland Hilman Fuadi, and Alham Fikri Aji. Predicting the order of upcoming tokens improves language modeling. *arXiv preprint arXiv:2508.19228*, 2025.

A TRAINING DATA MIX

Since we aim to improve math adaptation as a toy setting for our experiments, we upsample the math data compared to the data mix used in the final stage of pretraining of the base model SmolLM2 (Allal et al., 2025) from 14% to 40%, but maintain a strong representation (60%) of English web data to ensure that the model does not lose its general language generation capabilities. Specifically, for the math data we use the highest-quality 4-plus subset of the Nemotron-CC-Math dataset (Rabeeh Karimi Mahabadi, 2025; NVIDIA et al., 2025), which has been shown to outperform other math pretraining datasets on math reasoning benchmarks, including FineMath (Allal et al., 2025) which was used to train SmolLM2.

Our English web data is split evenly between DCLM-Edu (Allal et al., 2025) and FineWeb-Edu (Lozhkov et al., 2024), aligning closely with the 60/40 mix between these datasets used in the final stage of base model pretraining. However, for DCLM-Edu, we use a 3-plus subset (filtering to only include samples which achieved a score of 3+ using the FineWeb-Edu educational quality classifier). Meanwhile for FineWeb-Edu, we simply sampled data from the 100BT-sample subset.

The data was taken from the following datasets published to HuggingFace:

- `nvidia/Nemotron-CC-Math-v1` (***4plus*** subset)
- `HuggingFaceFW/fineweb-edu` (***sample-100BT*** subset)
- `semran1/dclm-edu-3-plus`

B TRAINING DETAILS

For majority of our experiments, we train SmolLM2 1.7B on 10B tokens (unless otherwise specified) using DDP on NVIDIA H200s, with a global batch size of $\sim$1.1M tokens (560 sequences of length 2048) and bfloat16 precision. We use a Warmup-Stable-Decay (WSD) learning rate scheduler to allow for variable training lengths, with 750 warmup steps, and reserving the final 10% of training for a cooldown to 0 with linear decay (Hägele et al., 2024). For example, in our 10B token experiments, the final $\sim$1B tokens (872 steps) are consumed during cooldown. We use a modest peak learning rate of 2.5×10^{-4}, which is half of the peak learning rate used in the original SmolLM2 pretraining, in order to avoid catastrophic forgetting and maintain the general capabilities of the model during adaptation (Ibrahim et al., 2024). Otherwise, we use the same optimizer parameters as SmolLM2 (AdamW optimizer with $\beta_1 = 0.9, \beta_2 = 0.95$ and weight decay of 0.01). We do not apply weight decay on normalisation layers, biases, nor the embedding layer (Bakouch et al., 2025). We also clip gradients to a max norm of 1.

For the experiments with the 360M parameter model, we use mostly the same experimental settings with a few changes. We use a global batch size of $\sim$570k tokens (280 sequences of length 2048), and peak learning rate of 6×10^{-4} ($0.2\times$ the original pretraining LR, since we find this is more stable) with warmup for 1000 steps. Additionally for LaLoR, we use $\ell_r = 8$ out of 32 layers, maintaining the 25% of layers for the *refine* stage. 360M models are trained using NVIDIA A100s.

C MTP BASELINES

A key difference of our MTP setup compared to MuToR (Gerontopoulos et al., 2025) is that they focus on a finetuning setting on prefix-answer tasks, and insert one register per regular token in the answer part of the sequence. As such, the number of tokens to learn from doubles in their setting (since during finetuning they do not calculate losses over prefix tokens). However, in order to maintain an equal token budget in our experiments compared to vanilla continual pretraining without register tokens, we keep the sequence length fixed and replace a proportion B_r of regular tokens with registers. This gives us an additional tradeoff, where we have to balance adding more MTP signal to get lower-variance estimates for the loss, versus maintaining enough regular tokens to not hurt the core learning. Accordingly, we also need to balance the loss contribution of the register tokens, since if we have high-variance estimates and a high weight, this could potentially hurt learning. Table 1 shows a small ablation on these two parameters when training SmolLM2 1.7B

Table 1: Ablation of MTP parameters B_r (register budget) and α (loss scaling) after training SmolLM2-1.7B for 10B tokens (with 1B token cooldown). The results for the pretrained base model are shown in gray, and the vanilla NTP training baseline is shown in the second row. We find that using a budget of 10% ($B_r = 0.1$) and α of 0.1 seems to offer the best improvement at this early stage.

		Math				Reasoning & Commonsense						Total
Register Budget	**α**	**GSM8K**	**Plus**	**MATH**	**Avg**	**ARC-E**	**ARC-C**	**Hella**	**PIQA**	**CQA**	**Avg**	**Avg**
		29.87%	16.79%	12.40%	19.69%	**77.78%**	47.27%	**71.43%**	77.42%	41.85%	63.15%	46.85%
—	—	33.28%	18.42%	14.80%	22.17%	77.36%	46.76%	70.64%	77.09%	42.26%	62.82%	47.58%
10%	0.1	33.74%	17.58%	**17.80%**	**23.04%**	77.48%	46.84%	70.48%	**77.53%**	**44.31%**	**63.33%**	**48.22%**
10%	0.2	**34.65%**	17.08%	15.80%	22.51%	77.48%	46.84%	70.50%	77.09%	43.65%	63.11%	47.89%
20%	0.1	33.89%	**18.67%**	14.60%	22.39%	77.23%	47.01%	70.41%	77.26%	42.83%	62.95%	47.74%

for 10B tokens. Using a small $B_r = 10\%$ budget along with $\alpha = 0.1$ seems to work well, so these are the parameters we use for all further experiments (unless otherwise specified).

The maximum prediction horizon d_{max} is also a key hyperparameter for MTP methods, however since multiple works such as Gloeckle et al. (2024) and Gerontopoulos et al. (2025) (both of which use very different MTP mechanisms) find that using up to 4^{th} token prediction seems to be effective in general, we use this value for the sake of simplicity. For now, our aim is not to optimize the MTP setup or achieve the strongest possible results; we simply need to fix reasonable parameters to enable us to do comparative analyses for our latent lookahead method. In future, we would revisit this parameter, particularly to study how we might be able to scale up the lookahead horizon.

D Evaluation Benchmarks

We group our evaluations into two categories: Math, and Reasoning & Commonsense.

Math Since our experimental setting involves trying to improve math adaptation by upsampling math data, our primary evaluation focuses on common math benchmarks. We report performance on: GSM8K (Cobbe et al., 2021) (1.3k test samples) with 5-shot examples, GSM-Plus (Li et al., 2024) with 5-shot examples to assess robustness against mathematical perturbations, and MATH (Hendrycks et al., 2021) for advanced mathematical reasoning. Specifically, we evaluate on the mini subset of GSM-Plus (2.4k samples), and the MATH500 subset (500 samples) for MATH using the Minerva 4-shot prompting format (Lewkowycz et al., 2022) and the Math-Verify evaluator (Kydlicek et al., 2025). We refer to these benchmarks as GSM8K, Plus and MATH respectively.

Reasoning & Commonsense We also monitor a range of standard reasoning and commonsense benchmarks, to measure the model's general capabilities and verify that the math upsampling does not cause catastrophic forgetting. We evaluate on the AI2 Reasoning Challenge (ARC) (Clark et al., 2018) (Easy and Challenge subsets), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), and CommonsenseQA (Talmor et al., 2019). We refer to these as ARC-E, ARC-C, Hella, PIQA and s respectively.

We report performances on individual tasks, the macro-averages within each group, and the macro-average across all tasks. Benchmarks are evaluated using LM Eval Harness (Gao et al., 2024) version 0.4.10dev0. Table 2 shows the configuration for each of our benchmarks, such as the number of few shot examples and the specific metric we report.

When evaluating with LaLoR, for generative tasks (in this case the three math tasks), we simply use the inference procedure outlined in 3.2.2. For tasks which measure the loglikelihood of given target sequences however (all the reasoning & commonsense benchmarks we test), we insert a set of registers with $d \in \{2, ..., d_{\text{max}}\}$ for each token in the target sequence (and discard these before calculating loglikelihoods), which produces equivalent outputs to the sequential inference process (if at each step we were to append the ground truth token).

Table 2: Configuration for benchmarks using LM Eval Harness

Task Category	Task Name	Number of Few-Shots	Score Metric/Filter
Math	GSM8K	5	strict-match
	GSM-Plus (Mini)	5	strict-match
	Minerva MATH500	4	math-verify
Reasoning & Commonsense	ARC-Easy	0	Acc
	ARC-Challenge	0	Acc Norm
	Hellaswag	0	Acc Norm
	PIQA	0	Acc Norm
	CommonsenseQA	0	Acc

E Inference Benchmarking

We perform ablations to measure the inference speeds when using LaLoR under different conditions. For our experiments, where not otherwise specified, we use a fixed prompt length S of 256 and generate 256 tokens per sequence. We compare our implementation of generation with refinement against the default HuggingFace generation function (using greedy decoding). We calculate the throughput (tok/s), latency (ms/tok) relative to the HF baseline, and the peak VRAM usage. We conduct all experiments using SmolLM2 1.7B on a single NVIDIA A100 GPU. For each condition, we run 2 batches as a warmup, and then average the results over the next 10 batches.

Figure 4 compares decoding speeds across batch sizes, Figure 5 tests different numbers of register tokens, whilst Figure 6 compares with different numbers of refinement layers ℓ_r. Overall, we see there is a consistent latency overhead across all experiments compared to the baseline, but under many conditions it is relatively minimal ($< 1.10\times$). In particular, for mid-range batch sizes, where we are GPU memory-bound (thus adding extra tokens is effectively free (Liu et al., 2025a)) but still not CPU-bound, the overhead is minimal. In the compute-bounded regime however with larger batch sizes, there is a significant drop off in performance with our method, since the additional computations required for lookahead refinement start to dominate.

It is important to note that our current implementation is not entirely optimized, and it should be possible to recover some of this latency. Our inference method requires the following overheads:

- Creating and maintaining an additional 4D attention mask
- Doing the forward pass with non-causal custom attention masks (i.e. not directly compatible with kernels optimized for causal attention)
- Switching the attention mask for the final ℓ_r *refine* layers
- Trimming the KV cache at each step to remove register tokens

Thus, there will always be some inference penalty to pay, however our current implementation is still comparable in latency to the HuggingFace default generation function, and there is scope for further optimization. For example, implementing a FlexAttention kernel (Dong et al., 2024) for faster attention calculations with our custom attention mask, or optimizing the cache trimming and additional attention mask creation (which is done relatively naively in Python for now). The mask switching is also performed using a model wrapper and pre-forward hooks which incur a small overhead (Figure 6), however there is likely a more optimized way to build this into the model directly.

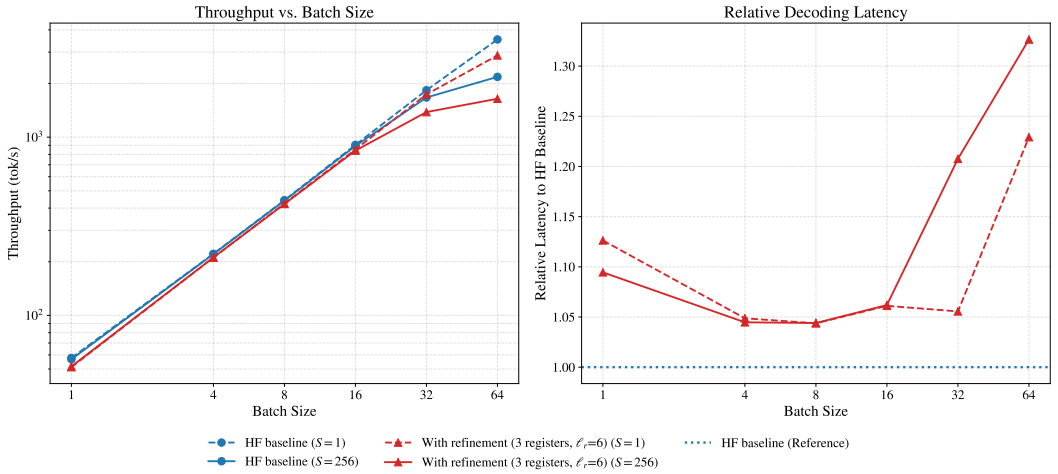


Figure 4: Throughput and decoding latency compared to the HuggingFace baseline when performing inference with different batch sizes, and prompt lengths of $S \in \{1, 256\}$. Latency spikes and throughput starts to degrade compared to the baseline at larger batch sizes (32+), likely due to the model moving into the compute-bound regime, whereby the additional register tokens result in a latency overhead. Meanwhile for smaller batches, inference is likely still memory-bound so the latency is relatively stable. There is however an initial spike in latency at batch size 1. This could be due to the overhead of the extra processing steps required for refinement (e.g. trimming the cache, maintaining an additional attention mask etc.), which could result in being CPU-bounded if the GPU is able to complete the batch size 1 computations quickly. Furthermore, when comparing a prompt length S of 1 compared to 256, we see that the larger prompt length degrades faster at higher batch sizes. Again, in the compute-bounded regime, this additional overhead could be due to the inefficient attention computations (which would be exacerbated with longer sequences) when using custom attention masks, compared to the optimized kernels HuggingFace can use by default when the inputs are purely causal.

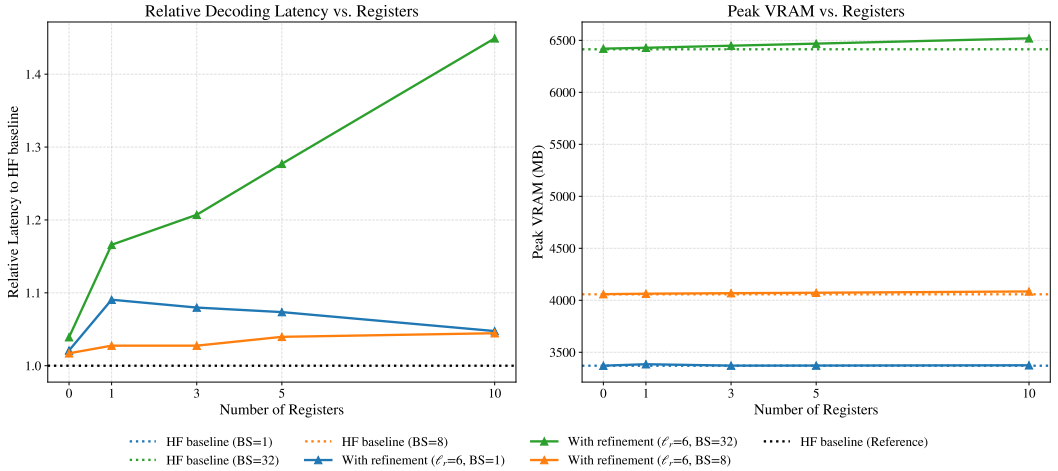


Figure 5: Decoding latency and peak VRAM usage relative to the HuggingFace baseline when doing inference with different number of registers and varying batch sizes (1, 8 and 32). At small batch sizes, whilst still in the memory-bounded regime, we are able to effectively scale the number of registers with minimal impact on latency, and only marginal increase in memory usage. Meanwhile, for a larger batch size of 32, the latency degrades almost linearly with the number of registers, likely due to the GPU now being saturated and thus moving into the compute-bound regime.

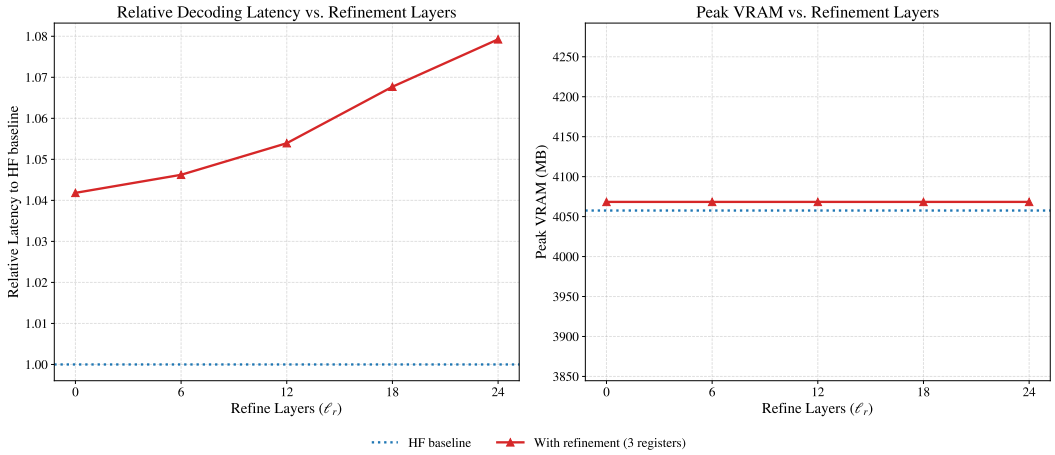


Figure 6: Decoding latency and peak VRAM usage relative to the HuggingFace baseline when doing inference with different number of *refine* layers ℓ_r (batch size 8). Increasing the number of refinement layers linearly increases latency, but has negligible effect on memory. This latency comes from additional pre-forward hooks to switch the attention mask in the *refine* layers. Latency with $\ell_r = 0$ is 18.93 ms/tok on average compared to 19.61 ms/tok for $\ell_r = 24$, which works out to a penalty of approximately 28 μs/tok per refinement layer on average.

F Attention Scores Analysis

Since our lookahead mechanism operates by adding bidirectional attention to allow the current token to attend to its registers, we perform some analyses to visualize the attention patterns to these future tokens.

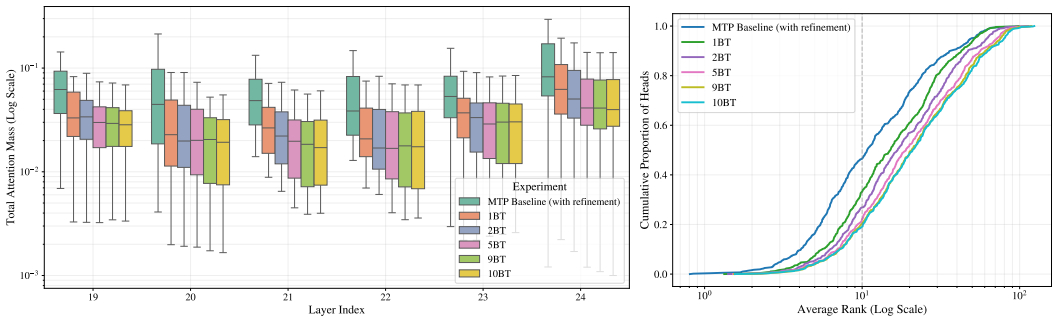
F.1 Analysis Method

We randomly sample 100 GSM8K test questions (prompt + answer), and append a set of $d_{\text{max}} - 1$ registers (in our case 3) with $d \in \{2, ..., d_{\text{max}}\}$ to each regular token in the answer part of the sequence. We then do a forward pass with $\ell_r (= 6)$ refinement layers, calculate attention scores of each answer token to each of its registers, and aggregate the average per-head attention scores for each refinement layer and each offset d. Since the significance of attention scores themselves can be hard to interpret, for every (regular) answer token, we also sort all tokens that it attends to by their attention score, and calculate the average rank of the registers at each offset. Using this, we visualize the following:

- How the attention scores (or attention mass, since we calculate after softmax) vary across refinement layers, and across heads within each layer, by constructing box plots of the distribution of the average register attention scores (across all offsets) per head within each refinement layer.

- How significant the attention to the register tokens is, by plotting a CDF of the rank of the attention to the register tokens within each head (across all layers).

F.2 BASELINE LALOR



(a) (b)

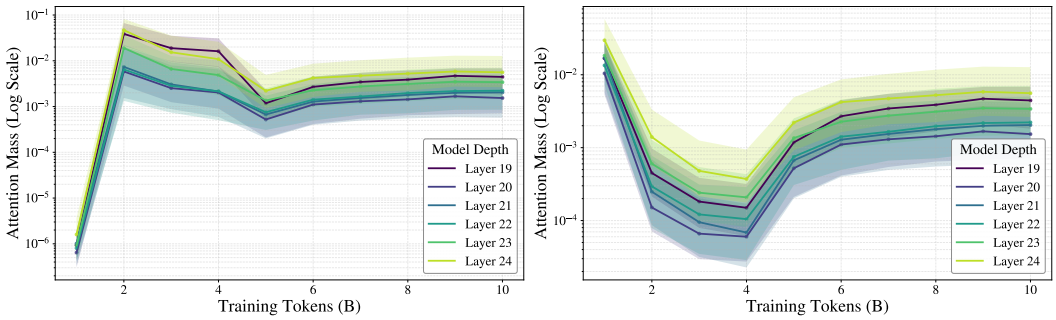
Figure 7: Analysis of the (lookahead) attention to register tokens at different stages of training SmolLM2 1.7B using LaLoR. The MTP Baseline (with refinement) result shows the attention behavior when enabling lookahead in the MTP baseline model which was trained without lookahead. This is used as a proxy to estimate a 'baseline' for the lookahead attention, to observe if the model learns to attenuate the attention or increase attention to registers compared to an 'untrained' lookahead mechanism. Plot (a) shows the distribution of attention scores to register tokens per head across layers. Plot (b) depicts the CDF of the rank of register attention scores within each head (see Appendix F.1 for details on how these are calculated). We observe that through training, the model learns to attenuate attention to the registers, strongly at first before stabilizing.

G ADDING A BIAS TO THE LOOKAHEAD ATTENTION

We perform additional experiments where we add an explicit bias to the attention mask, with the aim of encouraging the model to utilize the lookahead mechanism. Starting from an additive attention mask (applied before the softmax operation when calculating attention scores, with 0 and $-\infty$ values where attention should or should not be enabled respectively), we set the value of all new attention connections in the *refine* stage (in purple in Figure 1) to a bias value b.

We use a scheduler on the value of b to allow the following behavior. In the early stages of training, when the register tokens are still untrained and intermediate future representations are mostly noise, we set a large negative bias ($b = -10$), effectively disabling the refinement (for the first 1B tokens). Then, for the next 1B tokens, we linearly warm up the bias to a moderately high positive bias of $b = 5$, to explicitly encourage the model to look ahead. We remain with this positive bias for the next 2B tokens, before cooling down linearly to $b = 0$ over the course of another 1B tokens. Then for the rest of training, we remain with no bias, to allow the model to learn freely after hopefully learning that there is information to look ahead at.

Figure 8 shows the progression of the attention scores throughout training with this scheduled bias (with an analysis similar to that outlined in F.1).



(a) Inference with bias matching scheduler

(b) Inference with 0 bias

Figure 8: Progression of register token attention scores during training of a SmolLM2 1.7B mode using LaLoR with a scheduled attention bias. The plots show the median register attention scores (solid lines) and interquartile ranges (shaded regions) for attention heads in each layer of the *refine* stage. (a) shows the attention scores when we use the same bias at inference as the bias which would be active at the equivalent step in training, whilst (b) shows attention when performing inference with 0 bias always. Barring the first 1B token checkpoint, which has a high negative bias applied to effectively disable the refinement, for the rest of training, it seems that the model is actively trying to overcome the bias and maintain relatively stable attention scores. In (a), we see that at the onset of the high positive bias at 2B tokens, the attention scores are high, and over the following 2B tokens where this bias is maintained, the model is attenuating the bias. After cooling down the bias between 4B-5B tokens, the attention scores drop, and the model readjusts to increase attention until it seems to stabalise by the end of training. This indicates that the model is indeed likely learning how much to attend to registers based on its perceived utility of the registers. (b) also provides strong evidence that the model is not simply learning to ignore the register tokens, since if this was the case, after learning to diminish attention to the registers when accounting for the bias, it would likely not recover the attention scores as we remove the bias if there was no perceived utility for them.

H Changing the Register Insertion Strategy

Whilst in all other experiments we use the register token insertion strategy outlined in Section 3.2.1, with a fixed token budget and randomly inserted register tokens, here we test if relaxing this budget and changing the insertion strategy to something more structured can help. Specifically, we aim to address potential issues due to the sparsity of register tokens in the sequence, and the sparsity of register tokens within each lookahead interaction during training.

First, in regards to token budget, we no longer truncate any regular tokens. Instead, we simply add additional register tokens to the input. In practice, we add an equivalent number of registers as the input sequence length T, thus doubling our training sequence length. This of course significantly increases the computational and memory overhead during training, and in our experiments it slowed down throughput (in terms of regular tokens consumed) by $\sim 3\times$. However, we perform this experiment to verify that the sparsity itself is not the main bottleneck. In terms of training parameters, we keep these mostly the same. We keep the global batch size fixed in terms of number of sequences, thus we effectively double our batch size in terms of tokens. We also increase the α parameter for scaling the MTP loss to 0.3, since the original MuToR paper found this value to work best in general, and we are now closer to their register insertion regime which also doubles the learnable part of the sequence. This change in parameter seems to potentially hurt the MTP performance compared to the 0.1 value we use in all other experiments, however since we are most interested in a comparitive analysis between MTP and LaLoR, which will use the same α, this is not a major concern.

For the new register insertion strategy, we use mostly the same idea, however instead of sample tuples of N_r unique (t, d) pairs, where N_r is the number of register tokens to insert, we now only sample $\lfloor N_r/(d_{\text{max}} - 1) \rfloor$ positions t, and insert complete sets of $d_{\text{max}} - 1$ registers with $d \in \{2, ..., d_{\text{max}}\}$ at these positions. The results of these experiments are shown in Figure 9.

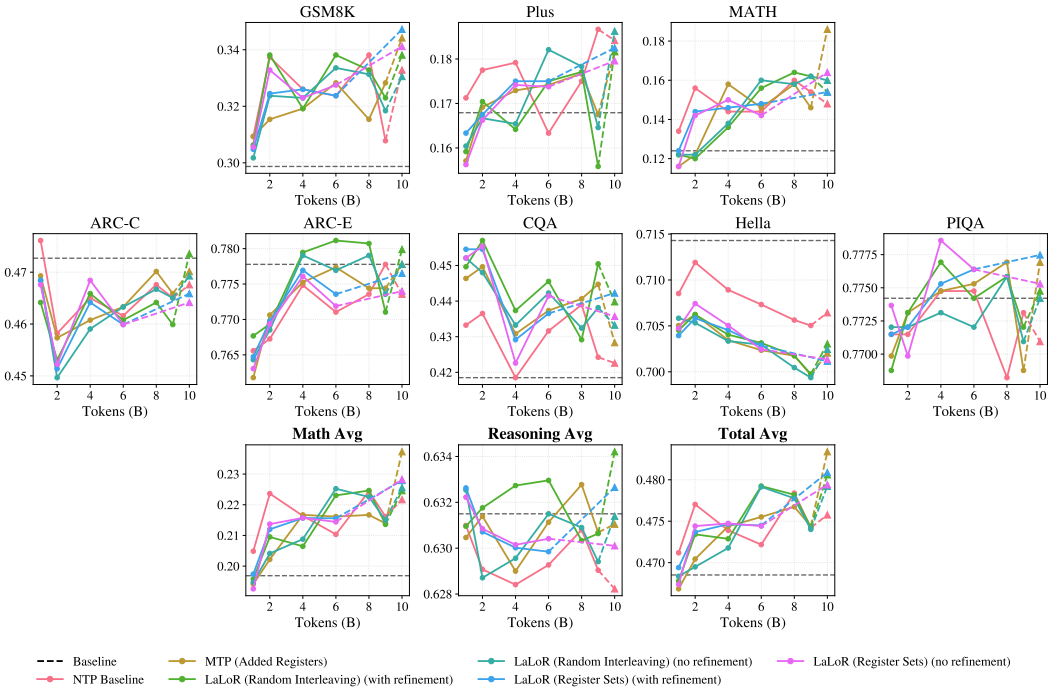


Figure 9: Evaluation results for SmolLM2 1.7B throughout the 10B token training using a additional registers rather than a constrained token budget. We additionally compare the original random strategy for interleaving register tokens into the input, with a modified insertion strategy which only includes complete sets of registers. Both strategies are tested in the added registers regime, and compared with a new MTP baseline which also uses additional registers. We find no significant performance differences when comparing with different interleaving strategies compared to the MTP baseline, and again no difference in performing inference with or without refinement, indicating that neither form of register sparsity is likely the main cause for refinement being ineffective.

I Adding an Intermediate Loss

I.1 Method

Since each register token $r_{t,d}$ is effectively trying to mimic the next-token prediction process of a future token x_{t+d-1}, we introduce a new loss to try to push the latent representations of the register tokens closer to that of the token they are imitating during the *refine* stage. More concretely, immediately after the *draft* stage, at the output of layer $L - \ell_r$, we calculate an intermediate loss $\mathcal{L}_{\text{Int}}$ as follows:

$$\mathcal{L}_{\text{Int}}(\theta) = \mathbb{E}_{\mathbf{x}' \sim \hat{\mathcal{D}}'} \left[\sum_{t=0}^{T} \sum_{d=2}^{d_{\max}} \mathbf{1}_{\{r_{t,d} \in \mathbf{x}'\}} \left(1 - S_C \left(\mathbf{h}_{r_{t,d}}^{(L-\ell_r)}, \mathbf{h}_{x_{t+d-1}}^{(L-\ell_r)} \right) \right) \right] , \tag{9}$$

where $S_C(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \|\mathbf{v}\|}$ is cosine similarity, and $\mathbf{h}_{r_{t,d}}^{(L-\ell_r)}$ and $\mathbf{h}_{x_{t+d-1}}^{(L-\ell_r)}$ are the hidden states of a register and the token it is imitating at the output of layer $L - \ell_r$. We detach gradients from the hidden state of the regular token to ensure only the registers learn from this additional loss. Our total loss then becomes

$$\mathcal{L}_{\text{Total}} = \mathcal{L}_{\text{MTP}} + \beta \mathcal{L}_{\text{Int}} , \tag{10}$$

where $\beta \geq 0$ is the weight for the intermediate loss. In our experiments, we use $\beta = 0.1$. For a fair comparison, we also run an additional experiment where we add this loss to standard MTP training without lookahead (calculated at the same layer).

I.2 RESULTS

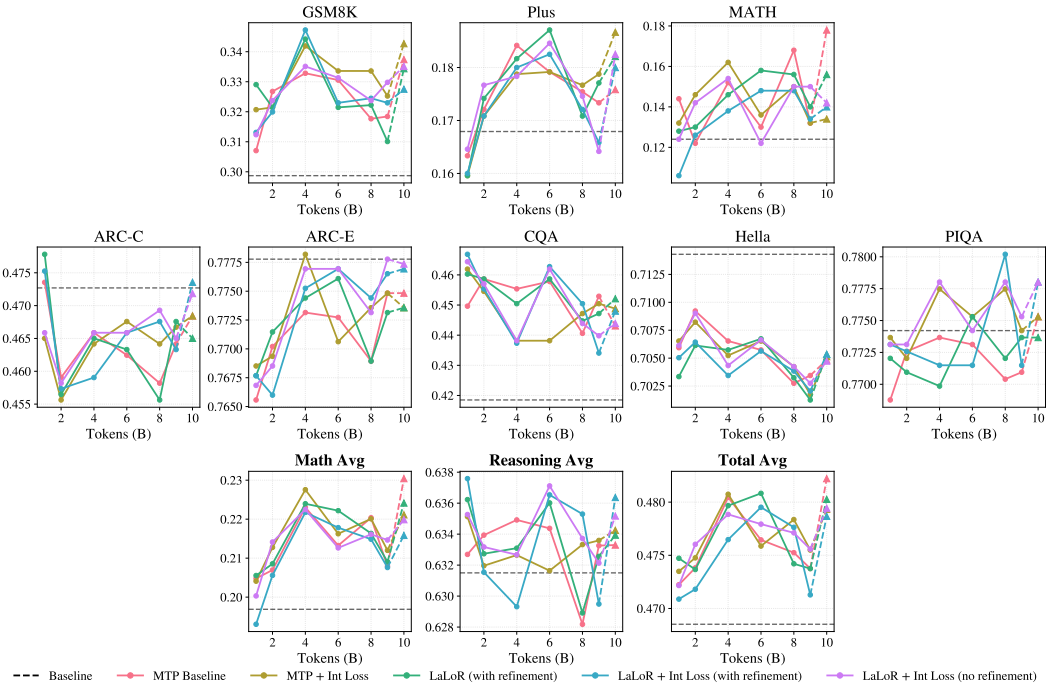


Figure 10: Evaluation results for SmolLM2 1.7B throughout the 10B token training using an additional intermediate loss with $\beta = 0.1$. The intermediate loss seems to make no significant difference in downstream performance for either the standard MTP training or LaLoR.

I.3 ANALYSING INFLUENCE ON HIDDEN STATES

We analyse the influence of this intermediate loss on the hidden representations of the register tokens, to measure if indeed they are pushed closer to the target representation of the regular token which has the equivalent next-token prediction target. For this analysis, we use a setup similar to what is described in Appendix F.1, however instead of aggregating attention scores, we calculate two metrics: the layer-wise cosine similarities of intermediate states, i.e. $S_C(\mathbf{h}^l_{r_{t,d}}, \mathbf{h}^l_{x_{t+d-1}})$; and the ratio of the L_2 norms $|\mathbf{h}^l_{r_{t,d}}|_2 / |\mathbf{h}^l_{x_{t+d-1}}|_2$ respectively for all $l \in \{1, ...L\}$ and $d \in \{2, ..., d_{\text{max}}\}$, averaged across all sequence positions t. Results of this analysis, comparing the LaLoR models trained with and without the intermediate loss, are shown in Figure 11. We see that the introduction of the intermediate loss does in fact seem to push the representations slightly closer for some intermediate layers (with this effect most pronounced at the layer in which the loss is introduced), however the representations in the final layers of the model are less affected by this loss.

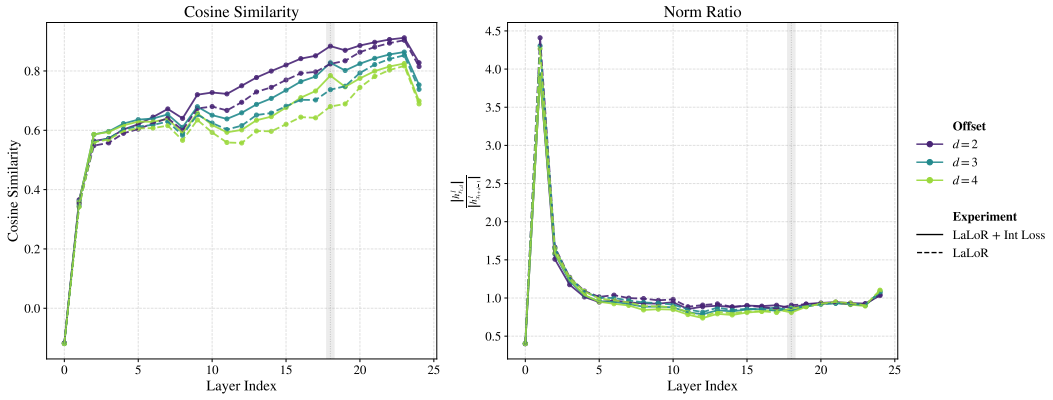


Figure 11: Comparison of layer-wise cosine similarity and norm ratios of intermediate representations of register tokens (at each offset) compared to the regular token they are imitating (i.e. with the same prediction target) when training with LaLoR with and without the intermediate loss. The highlighted layer indicates the layer at which this loss was calculated. We see that the intermediate representations do get somewhat closer to the target representation in the middle layers of the network, with a spike in improvement at the layer where the loss is calculated. However, the effect on the final layers of the model is less pronounced.

I.4 Analysing Influence on Attention

Figure 12 shows how the lookahead attention scores change across the *refine* layers when introducing this intermediate loss (using the same analysis setup as F.1). We see that the intermediate loss seems to increase attention to the registers. Comparing to Figure 11, we also observe a high correlation between the improvement in hidden representation and increase in attention scores. Layers closer to the start of the *refine* stage where the loss is calculated seem to benefit more in terms of representation and also gain higher attention scores, whilst the effect in both analyses is minimal for the final layers of the model.

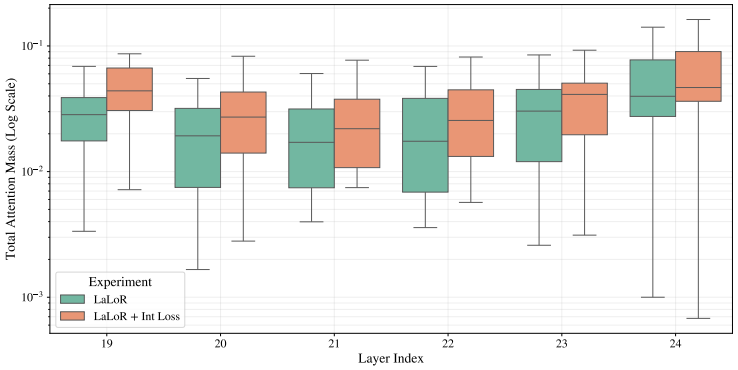


Figure 12: Distribution of the attention scores to register tokens in each head, across all layers in the *refine* stage, for SmolLM2 1.7B models trained using LaLoR with and without an intermediate loss. We see that the attention scores when using an intermediate loss increase compared to training without, particularly in the earlier layers of the *refine* stage closer to where the intermediate loss is calculated.

J 360M Parameter Model Results

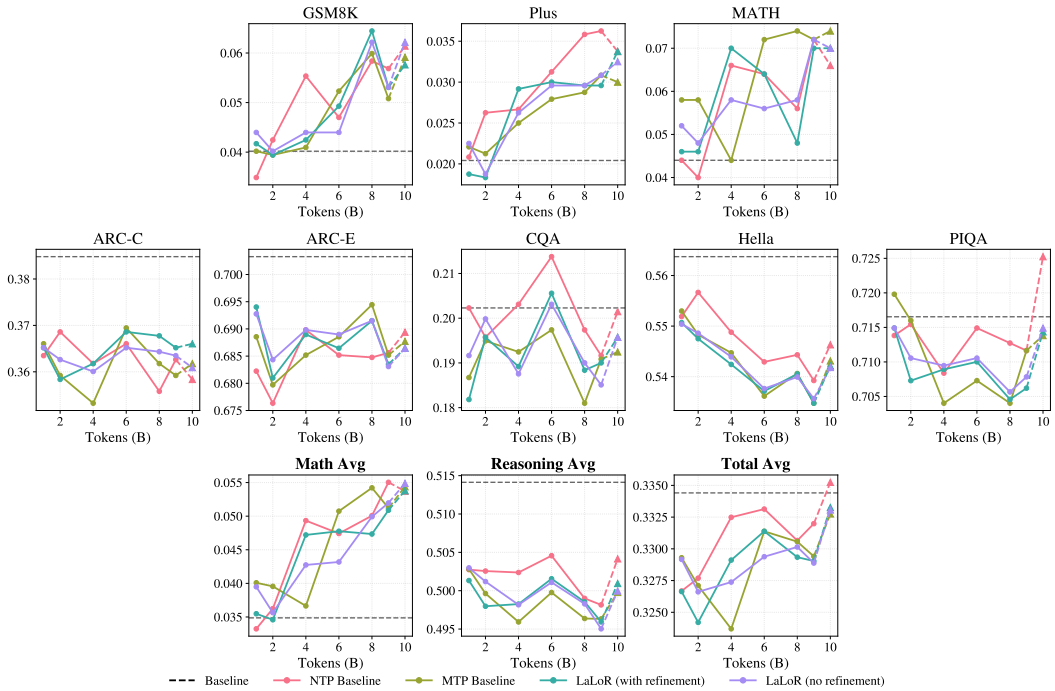
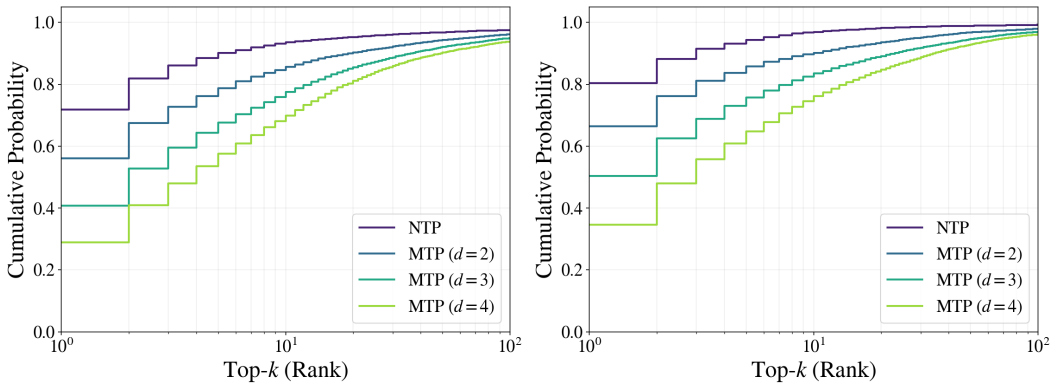


Figure 13: Evaluation results for SmolLM2 360M throughout the 10B token training. Whilst all methods improve on math benchmarks, the continual training hurts reasoning & commonsense capabilities. Comparing between methods, there is not much differentiation, and again evaluations at different checkpoints are relatively noisy. However, it seems that whilst methods using an MTP objective (including LaLoR) perform very similarly across tasks, NTP performs marginally better on average in reasoning & commonsense tasks, though this is not conclusive.



(a) SmolLM2 360M (b) SmolLM2 1.7B

Figure 14: CDF of the rank of the target token in the predictions by regular (NTP) tokens, as well as register (MTP) tokens at different offsets d, for the 360M (a) and 1.7B (b) parameter MTP Baseline models. We see that the top-k accuracies of MTP with the larger 1.7B model is consistently higher than the 360M model, suggesting the MTP performance can likely improve with scale. Additionally, although not a perfect proxy for acceptance rates in speculative decoding, the non-zero top-1 accuracies of the MTP suggests we can gain at least some inference speedups when applying a speculative decoding scheme atop MTP (or LaLoR).