PARALLEL LANGUAGE MODELS VIA VARIATIONAL DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive language models emit one token per forward pass. This causes high inference latency, in particular as models are scaled up for quality. In this paper, we introduce Parallel Language Models (PLMs), which generate mutually consistent tokens in one forward pass by routing dependencies through latent auxiliary variables. PLMs are trained using a variational bound that reduces to an autoregressive loss plus a parallel decoding loss. Empirically, PLMs are on a faster and better Pareto frontier than autoregressive and discrete diffusion baselines. Pretrained PLMs achieve near-autoregressive pretraining loss on model scales of 80M to 3B parameters, and speedup follows a power law in model size. When post-trained, our 1.3B model matches or exceeds an identically trained autoregressive model on math, coding, and instruction-following benchmarks, speeding up inference 2.3-5.7$\times$. Together, we show that parallel generation can be trained from a single training objective at autoregressive quality.

1 INTRODUCTION

Large autoregressive models (Vaswani et al., 2017) are the core technical foundation of large language models (LLMs) (Brown et al., 2020). Autoregressivity comes with a significant downside, however: One model forward pass produces one token, causing high inference latency.

A plethora of approaches address this challenge, most prominently by increasing the speed of a given autoregressive model through smaller draft models in a process known as speculative decoding (Leviathan et al., 2023). However, draft models do not reach the quality of their big counterparts, ultimately limiting the achievable speedup. Discrete diffusion models (Hoogeboom et al., 2021; Austin et al., 2021) are an alternative modeling framework that predict several tokens in one model call. However, a single model call cannot coordinate predicted tokens, requiring iterative refinement. Few-step discrete diffusion models reduce the number of steps by amortizing iterative refinement (Monsefi et al., 2026). Because diffusion models are more expensive to train, autoregressive models still dominate industry-scale inference.

This calls for a language modeling framework rooted in autoregression that can model arbitrary dependencies between predicted tokens in a single model call, combining effective training with fast inference at equal quality. Recently, Parallel Token Prediction (PTP) presented a decoding framework that distills an existing autoregressive model into a fast decoder, predicting several tokens in parallel (Draxler et al., 2026). However, PTP was only trained as a speculative decoding method for an existing model, and the framework they proposed to train from sequence data alone is highly inefficient because it is iterative at training time.

In this work, we close this gap and propose Parallel Language Models (PLMs), a novel language modeling framework based on an autoregressive architecture that trains parallel LLMs directly from training data. In detail, we contribute:

- We build Parallel Language Models based on a variational formulation, deriving effective pretraining and post-training strategies (Section 3).
- We pretrain PLMs up to 3B parameters that are on a faster and better Pareto frontier than autoregressive and discrete diffusion baselines (Fig. 1). Our scaling laws confirm near-autoregressive negative log-likelihoods, and show that the parallelism increases with model scale (Section 4.1.1).

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

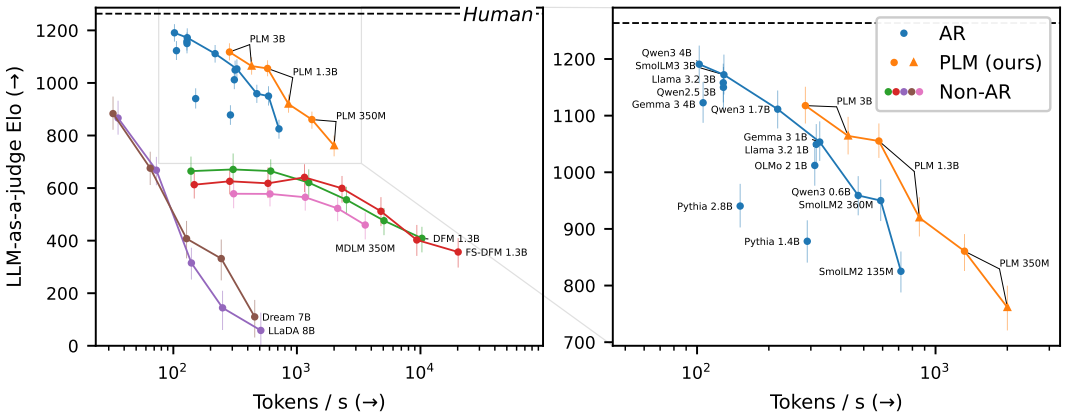

Figure 1: **Parallel Language Models (PLMs, orange) shift the Pareto frontier of speed vs quality for language models trained from scratch.** *(Right)* At the same inference speed, PLMs outperform the quality of publicly available autoregressive models (AR, blue), despite the latter being trained on trillions of tokens and our models only trained on 25B (PLM 350M) to 300B tokens (PLM 3B). *(Left)* PLMs outperform discrete diffusion approaches at various NFEs. Inference speeds using `vLLM` on NVIDIA RTX 6000 Ada. PLM circle and triangle refer to `exact` respectively `relaxed` inference, see Section 3.4. Error bars are 95% confidence intervals.

- When post-trained, our 1.3B and 3B Parallel Language Models match or exceed autoregressive quality while answering 2.3-5.7× faster (Section 4.2).

2 Related Work

Parallel Prediction and Generation. Early non-autoregressive sequence models generate multiple output tokens in parallel, substantially reducing the number of sequential decoding steps (Gu et al., 2018). More recently, multi-token prediction (MTP) predicts several future positions (Gloeckle et al., 2024) and is used in large-scale autoregressive models such as DeepSeek-V3 (DeepSeek-AI et al., 2025). These methods retain an autoregressive modeling framework rather than defining a joint parallel generative model. Parallel Token Prediction (PTP) (Draxler et al., 2026) instead introduces auxiliary random variables that enable multiple dependent future tokens to be generated jointly in one model call, and studies distillation from existing autoregressive models. Samragh et al. (2025) improve coherence of independent multi-token prediction by adding a small autoregressive head, and introduce Quadratic Decoding, which we adapt for PLM inference. Our work extends this line of research to pretraining of parallel language models directly from sequence data, developing a training recipe that scales to billions of parameters.

Iterative and Few-Step Parallel Generation. A complementary class of methods predicts many token positions in parallel and resolves dependencies through iterative refinement. Conditional Masked Language Models (Ghazvininejad et al., 2019) progressively fill and revise masked positions, while discrete diffusion language models such as MDLM (Sahoo et al., 2024), LLaDA (Nie et al., 2025), and Dream (Ye et al., 2025) generate sequences through repeated denoising. Discrete Flow Matching (Gat et al., 2024) provides a related framework for defining generative trajectories over discrete sequences, while Eso-LM (Sahoo et al., 2026) connects autoregressive and masked-diffusion modeling and improves diffusion inference. Recent work increasingly targets few-step generation: Infinite Mask Diffusion (Yoo et al., 2026) and FS-DFM (Monsefi et al., 2026) modify the generative process or training objective to improve performance under a small number of sampling steps, while flow-map approaches further compress iterative trajectories toward few- or single-step generation (Lee et al., 2026; Potaptchik et al., 2026). Relatedly, Lookahead Decoding (Fu et al., 2024) and Jacobi Forcing (Hu et al., 2026) generate tentative future tokens in parallel and recover consistency through verification or iterative refinement. In contrast, PLMs aim to model dependencies among simultaneously generated tokens directly within a single model call.

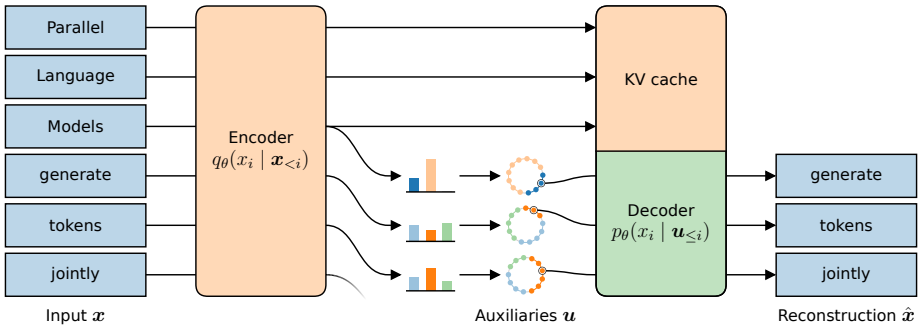

Figure 2: Parallel Language Models jointly train an autoregressive model that maps training sequences $\boldsymbol{x}$ to auxiliaries $\boldsymbol{u}$, and a decoder that reconstructs them back to sequences $\hat{\boldsymbol{x}}$ in parallel.

Autoregressive Inference Acceleration. A separate line of work accelerates inference from pretrained autoregressive language models. Speculative decoding uses a cheaper draft model to propose multiple tokens, which a target model verifies in parallel (Leviathan et al., 2023; Chen et al., 2023). Subsequent methods improve proposal quality through learned prediction heads or feature-based drafting, including Medusa (Cai et al., 2024), Hydra (Ankner et al., 2024), and EAGLE (Li et al., 2024a;b; 2025b). Unlike PLMs, these methods accelerate inference from a pre-existing autoregressive target model rather than training a parallel generative model directly from sequence data.

Latent-Variable Sequence Models. Latent-variable models can also capture dependencies while retaining parallel sequence generation. FlowSeq (Ma et al., 2019) models non-autoregressive sequence generation through a flow-based latent distribution, while LaNMT (Shu et al., 2020) uses continuous latent variables and variational training for non-autoregressive translation. Parallel WaveNet (Oord et al., 2018) similarly generates audio in a single pass by mapping per-timestep noise through an inverse autoregressive flow, but requires invertibility. These approaches demonstrate how latent representations can mitigate the dependency limitations of factorized parallel decoders, but primarily target conditional sequence-to-sequence or audio generation. PLMs instead use token-level auxiliary variables to define a general language model in which dependencies among parallel predictions are mediated through the latent representation.

3 Parallel Language Models

Autoregressive language models represent the distribution of language by explicitly predicting a histogram over possible choices from a token vocabulary $\{1, \ldots, V\}$. This is an expressive paradigm, but it does not extend to predicting several tokens jointly because the possible token combinations V^L scale exponentially in sequence length L.

One effective way to circumvent this is to use a language model that predicts sequences directly. Here, we sample a random auxiliary $u_i \sim p(u_i)$ per token position $i = 1, \ldots, L$, and then have a decoder map the vector of auxiliaries $\boldsymbol{u}$ to a coherent sequence in one model call.

$$\boldsymbol{x} = f_\theta(\boldsymbol{u}) \in \{1, \ldots, V\}^L. \tag{1}$$

Intuitively, the model generates different sequences for different random inputs. Prominent examples of such parallel decoders are Few-Step Discrete Flow Matching (FS-DFM) (Monsefi et al., 2026) and Parallel Token Prediction (PTP) (Draxler et al., 2026). Both frameworks distill a parallel decoder from a pre-existing model, the former from a discrete diffusion model, and the latter from an autoregressive language model.

In the remainder of this section, we discuss the core framework of Parallel Language Models (PLMs). PLMs directly train on sequence data without relying on a pre-existing model. We provide a complete scalable recipe, covering pretraining, post-training, and inference.

3.1 Self-Distilling Parallel Language Models on Sequence Data

The Parallel Language Model we propose is a self-distillation method that jointly trains an autoregressive encoder $q_\theta(u_i \mid \boldsymbol{x}_{\leq i})$ from sequences to auxiliaries, and a parallel decoder from auxiliaries to sequences $x_i = f_\theta(\boldsymbol{u}_{\leq i})$, see Fig. 2. Note that for notational simplicity, we remove dependency on a prompt or partially generated text.

To train it, we derive our training objective from maximizing the likelihood of the training data under our model:

$$\min_\theta -\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}}[\log p_\theta(\boldsymbol{x})]. \tag{2}$$

What is the probability of generating a sequence $\boldsymbol{x} = \boldsymbol{x}_{1:L}$ using a parallel decoder $x_i = f_\theta(\boldsymbol{u}_{\leq i})$? It is given by integrating over all possible auxiliaries from their prescribed distribution $u_i \sim p(u_i)$, and then decoding them to the corresponding sequence:

$$\log p_\theta(\boldsymbol{x}) = \log \int \prod_{i=1}^{L} p(u_i)\, p_\theta(x_i \mid \boldsymbol{u}_{\leq i})\, d\boldsymbol{u}. \tag{3}$$

Note how we have switched from a deterministic decoder f_θ to a stochastic decoder p_θ. This will allow taking gradients easily through a cross-entropy loss, and we take an $\arg\max$ for inference (see Section 3.4). Crucially, the product distribution over future tokens ensures that we can predict tokens in parallel: All dependencies between tokens are described by the auxiliaries.

Optimizing the loss in Eq. (2) using the integral in Eq. (3) is intractable: if we were to approximate the integral using a finite number of random auxiliary samples, we would essentially never sample a given sequence even once. This is where the encoder $q_\theta(\boldsymbol{u} \mid \boldsymbol{x}) = \prod_{i=1}^{L} q_\theta(u_i \mid \boldsymbol{x}_{\leq i})$ enters, proposing auxiliaries that should generate a given sequence. We rewrite Eq. (2) using the evidence lower bound (ELBO) (Kingma & Welling, 2013), see Appendix A.1 for details:

$$-\log p_\theta(\boldsymbol{x}) \leq \mathbb{E}_{\boldsymbol{u}\sim q_\theta(\boldsymbol{u}\mid\boldsymbol{x})}\left[\sum_{i=1}^{L} -\log p_\theta(x_i \mid \boldsymbol{u}_{\leq i})\right] + \sum_{i=1}^{L} D_{\text{KL}}(q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) \parallel p(u_i)). \tag{4}$$

Here, $p(u_i)$ specifies the distribution we sample the auxiliaries from during inference.

Equation (4) is significantly easier to optimize than Eq. (3), since the encoder proposes auxiliaries $\boldsymbol{u}$ for decoding a given sequence $\boldsymbol{x}$. We show in Appendix A.2 that equality holds at the global minimum, and that this global minimum can be achieved by a rich enough model, and so optimizing Eq. (4) actually optimizes Eq. (2).

Design choices We leverage the strong empirical performance of next-token prediction by factorizing our encoder via $q_\theta(\boldsymbol{x}, \boldsymbol{u}) = \prod_{i=1}^{L} q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) \prod_{i=1}^{L} q_\theta(x_i \mid \boldsymbol{x}_{<i})$. That is, our encoder first predicts the likelihood of the next token and then chooses an appropriate auxiliary u_i based on the observed token x_i. With this choice, and additional injectivity and regularity constraints, we can rewrite the encoder KL divergence from Eq. (4):

$$D_{\text{KL}}(q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) \parallel p(u_i)) = -\log q_\theta(x_i \mid \boldsymbol{x}_{<i}), \tag{5}$$

exactly the standard negative autoregressive model loss (derivation in Appendix A.3).

In addition, we decide not to compute any gradients through the sampling of $\boldsymbol{u}$, as passing gradients causes unstable training for us and the optimum does not change.

PLM Training Objective Together, we obtain the following remarkably simple training objective for a Parallel Language Model:

$$\mathcal{L}(\theta; \boldsymbol{x}) = \underbrace{\mathbb{E}_{\boldsymbol{u}\sim q_\theta(\boldsymbol{u}\mid\boldsymbol{x})}\left[\sum_{i=1}^{L} -\log p_\theta(x_i \mid \boldsymbol{u}_{\leq i})\right]}_{\mathcal{L}_{\text{dec}}(\theta;\boldsymbol{x})} + \underbrace{\sum_{i=1}^{L} -\log q_\theta(x_i \mid \boldsymbol{x}_{<i})}_{\mathcal{L}_{\text{enc}}(\theta;\boldsymbol{x})}. \tag{6}$$

It composes an autoregressive encoder with a parallel decoder. The above derivation confirms that this is a consistent way to train a parallel model. Next, we discuss efficient implementation.

3.2 Scalable Training

For a given training sequence $\boldsymbol{x}_{1:L}$, the PLM encoder q_θ first encodes every token x_i in a sequence to its auxiliary u_i, and then the decoder p_θ reconstructs it to $\hat{x}_i$, see Fig. 2. We now discuss practical design choices for training a Parallel Language Model that scales to billions of parameters and tokens.

Weight-sharing Encoder and Decoder Since a majority of the difficulty in predicting language is in parsing context, we share the model weights between the encoder q_θ and the decoder p_θ. They only differ in the embedding layer, which embeds tokens from the vocabulary differently from the auxiliaries.

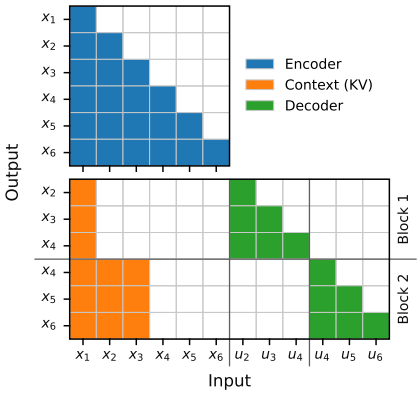

Figure 3: *(Top)* The encoder q_θ has a standard causal attention mask (blue). *(Bottom)* The decoder p_θ attends to the input tokens before its block. For example, the first block attends to the first input token x_1 via the encoder (orange), and predicts $\boldsymbol{x}_{2:4}$ at the positions receiving auxiliaries $\boldsymbol{u}_{2:4}$ (green).

Block-autoregressive Decoding Empirically, we find that reconstruction becomes increasingly difficult with sequence length. Therefore, we cap the maximum decoding sequence length to a block size K. For a training sequence of length $L > K$, we divide the input sequence $\boldsymbol{x}$ into blocks of K tokens. So to predict a block of tokens $\hat{\boldsymbol{x}}_{i_b:i_b+K-1}$ starting at position i_b, the decoder receives the corresponding auxiliaries $\boldsymbol{u}_{i_b:i_b+K-1}$ and the input tokens $\boldsymbol{x}_{<i_b}$ before the block. To avoid recomputation, we let the decoder attend to the KV cache of the encoder. Figure 3 shows how this can be efficiently implemented with an attention mask that makes each decoder block attend to the encoder tokens up to the block start.

Encoder-to-decoder Ratio While the optimum of Eq. (6) can encode and decode perfectly at infinite model capacity, in practice model capacity is split between the two modes of the model. We therefore introduce a loss ratio β, analogous to the loss weight in a β-VAE (Higgins et al., 2017). We implement this ratio by not decoding all tokens that were encoded ($\beta > 1$), or by decoding every token several times from multiple auxiliaries ($\beta < 1$). For $\beta = 1$, every encoded token is decoded once. To this end, we use a stride $s = \lfloor \beta K \rfloor$ between blocks. Figure 3 shows an example for stride $s = 2$ at block size $K = 3$. The explicit loss becomes:

$$\mathcal{L}(\theta; \boldsymbol{x}) = \underbrace{\sum_{b=1}^{B} \sum_{k=0}^{K-1} -\log p_\theta\left(x_{i_b+k} \mid \boldsymbol{x}_{<i_b}, \boldsymbol{u}^{(b)}_{i_b:i_b+k}\right)}_{\mathcal{L}_{\text{dec}}(\theta;\boldsymbol{x})} + \underbrace{\sum_{i=1}^{L} -\log q_\theta(x_i \mid \boldsymbol{x}_{<i})}_{\mathcal{L}_{\text{enc}}(\theta;\boldsymbol{x})}. \tag{7}$$

We denote models by PLM-β; the autoregressive baseline is recovered in the limit $\beta \to \infty$.

Geometry-aware Auxiliaries So far, we have left open how the encoder turns a token x_i into an auxiliary u_i, given the histogram $q_\theta(x_i \mid \boldsymbol{x}_{<i})$. We use a fixed pool of A auxiliaries that is divided among the tokens that could occur at position i. Each candidate token v receives a disjoint share of the pool proportional to its probability $q_\theta(x_i = v \mid \boldsymbol{x}_{<i})$; we call this share its *cell* $\mathcal{C}_v$. The observed token x_i is encoded by an auxiliary drawn uniformly from $\mathcal{C}_{x_i}$, and since the cells do not overlap, every auxiliary decodes to exactly one token. An auxiliary drawn uniformly from the whole pool therefore decodes to token v with exactly the probability $q_\theta(x_i = v \mid \boldsymbol{x}_{<i})$. Figure 4 illustrates this mapping.

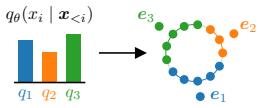

Figure 4: Auxiliaries are grouped around token embeddings e_v, with group sizes matching token probabilities.

Any division with these shares is valid; since the decoder has to recover x_i from u_i, we choose one that makes this as easy as possible. PTP (Draxler et al., 2026) uses the quantile intervals of $q_\theta(x_i \mid \boldsymbol{x}_{<i})$ as cells: an auxiliary is a number in $[0, 1]$, and to decode it the decoder must know the cumulative probability of all tokens with a smaller vocabulary index, an order that carries little meaning. We instead divide the pool in the model's own embedding space.

The auxiliaries are A extra tokens $\mathcal{U} = \{V+1, \ldots, V+A\}$ appended to the vocabulary, with fixed random unit vectors $\boldsymbol{e}_u \in \mathbb{S}^{d-1}$ as their embeddings and a uniform prior $p(u) = \mathrm{Unif}(\mathcal{U})$. Each candidate token v claims the auxiliaries whose embeddings lie closest to its own output embedding $\boldsymbol{e}_v$, up to a per-token offset w_v that sets how many it may claim: raising w_v grows the cell of v at the expense of its neighbors,

$$\mathcal{C}_v = \Big\{ u \in \mathcal{U} : v = \arg\max_{v'} \big(\boldsymbol{e}_{v'} \cdot \boldsymbol{e}_u + w_{v'} \big) \Big\}. \tag{8}$$

This is a weighted Voronoi diagram (Aurenhammer, 1987) on the sphere. At every position we fit the offsets so that the cells hold their prescribed shares up to rounding,

$$|\mathcal{C}_v| \approx A \, q_\theta(x_i = v \mid \boldsymbol{x}_{<i}), \tag{9}$$

such that Eq. (5) holds approximately, see Appendix B. In practice we restrict the candidates to the M most likely tokens and renormalize q_θ over them; an observed token outside this set reads a learned mask embedding instead (Appendix C). The payoff is that the auxiliary the decoder receives was chosen for its proximity to $\boldsymbol{e}_{x_i}$, so the decoder can read the intended token off the geometry rather than reconstruct a position in the vocabulary order. We find that this geometry-aware encoding outperforms the quantile encoding of PTP, see Appendix D.

Representation Alignment Each training step looks at the same token twice: once autoregressively, and once at parallel decoding. We find that adding a representation alignment loss (Yu et al., 2025) that aligns the hidden state of the parallel decoder to the autoregressive encoder significantly speeds up training, achieving the same decoding quality at approximately half the training compute Appendix D. We use the following regularization:

$$\mathcal{L}_{\text{align}}(\theta, \psi; \boldsymbol{x}) = \sum_{b=1}^{B} \sum_{k=0}^{K-1} \left(1 - \frac{g_\psi\big(\boldsymbol{h}^{(b)}_{i_b+k}\big) \cdot \mathrm{sg}\big[\boldsymbol{h}_{i_b+k}\big]}{\big\| g_\psi\big(\boldsymbol{h}^{(b)}_{i_b+k}\big) \big\|_2 \, \big\| \mathrm{sg}\big[\boldsymbol{h}_{i_b+k}\big] \big\|_2} \right). \tag{10}$$

Here, $\boldsymbol{h}^{(b)}_i$ is the hidden state the parallel decoder holds at position i of block b, read at layer ℓ_{dec}, and $\boldsymbol{h}_i$ is the autoregressive state at the same position, read at a deeper layer $\ell_{\text{enc}} > \ell_{\text{dec}}$. This asks the decoder to arrive early at what the encoder reaches later. The stop gradient $\mathrm{sg}[\cdot]$ makes the decoder chase the encoder and never the reverse, and the learned linear map g_ψ facilitates alignment. We add $\lambda_{\text{align}} \mathcal{L}_{\text{align}}$ to Eq. (7) and drop g_ψ after training, so inference is unchanged.

3.3 Post-training

We post-train PLMs in two stages. First, supervised fine-tuning (SFT) adapts the model to instruction–response data using the joint encoder–decoder objective of Eq. (7). We train on a mixture of general instruction-following, mathematics, coding, and precise instruction-following data. Dataset and optimization details are given in Appendix F.1.

We then apply Direct Preference Optimization (DPO) (Rafailov et al., 2023) jointly to the two modes of the PLM: the autoregressive encoder and the parallel decoder. Let θ denote their shared parameters and θ_{ref} a frozen SFT reference. For a prompt–response sequence $\boldsymbol{x} = (\boldsymbol{p}, \boldsymbol{r})$, let $\mathcal{L}^{\boldsymbol{r}}_{\text{enc}}(\theta; \boldsymbol{x})$ and $\mathcal{L}^{\boldsymbol{r}}_{\text{dec}}(\theta; \boldsymbol{x}, \boldsymbol{u})$ denote the response-restricted encoder and decoder components of Eq. (7). We define the response score

$$s_\theta(\boldsymbol{x}; \boldsymbol{u}) = -\frac{1}{L^{\boldsymbol{r}}(1 + 1/\beta)} \big(\mathcal{L}^{\boldsymbol{r}}_{\text{enc}}(\theta; \boldsymbol{x}) + \mathcal{L}^{\boldsymbol{r}}_{\text{dec}}(\theta; \boldsymbol{x}, \boldsymbol{u}) \big). \tag{11}$$

The fraction normalizes by the total number of terms, preventing the response length $L^{\boldsymbol{r}}$ from dominating the score.

For each candidate, we sample auxiliaries from the frozen reference encoder, $\boldsymbol{u}^{\pm} \sim q_{\theta_{\text{ref}}}(\boldsymbol{u} \mid \boldsymbol{p}, \boldsymbol{r}^{\pm})$, where each auxiliary is *target-compatible*: it is drawn from the observed target token's Voronoi cell. The same candidate-specific auxiliaries are used to evaluate the policy and reference, keeping the reference-relative comparison conditioned on a fixed latent realization.

Define $\delta_\theta(\boldsymbol{p}, \boldsymbol{r}; \boldsymbol{u}) = s_\theta(\boldsymbol{p}, \boldsymbol{r}; \boldsymbol{u}) - s_{\theta_{\text{ref}}}(\boldsymbol{p}, \boldsymbol{r}; \boldsymbol{u})$. For preference triples $(\boldsymbol{p}, \boldsymbol{r}^+, \boldsymbol{r}^-)$, we optimize

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{\substack{(\boldsymbol{p},\boldsymbol{r}^+,\boldsymbol{r}^-)\sim\mathcal{D}_{\text{pref}} \\ \boldsymbol{u}^+\sim q_{\theta_{\text{ref}}}(\cdot|\boldsymbol{p},\boldsymbol{r}^+) \\ \boldsymbol{u}^-\sim q_{\theta_{\text{ref}}}(\cdot|\boldsymbol{p},\boldsymbol{r}^-)}} \log\sigma\left(\gamma\left[\delta_\theta(\boldsymbol{p}, \boldsymbol{r}^+; \boldsymbol{u}^+) - \delta_\theta(\boldsymbol{p}, \boldsymbol{r}^-; \boldsymbol{u}^-)\right]\right), \tag{12}$$

where γ is the DPO inverse temperature. For the autoregressive baseline, we use the same objective with the decoder component omitted from Eq. (11).

Unlike canonical DPO, which uses summed sequence log-probabilities, our score is normalized by the number of prediction terms, similar to the length-normalized DPO of Lambert et al. (2025). SimPO (Meng et al., 2024) uses the same normalization but drops the reference model, which we keep. Further details are given in Appendix F.2.

3.4 Inference

During inference, tokens are generated in parallel via Quadratic Decoding (Samragh et al., 2025), whose two main components, proposal and verification, we adapt for PLMs. First, a block of K coordinated tokens $x_j = f_\theta(\boldsymbol{x}_{<i}, \boldsymbol{u}_{i..j})$, $i \leq j < i + K$ is produced by feeding independently randomly drawn auxiliaries $\boldsymbol{u}_j \sim p(u_j)$ (cf. Section 3.2) into our trained model p_θ and then taking the $\arg\max$ of the produced logits. Second, the proposed tokens are verified against a specified rule and a subset of accepted tokens is appended to the context. While our model has no theoretical limit on how many coordinated tokens it can produce, this step is necessary, as in practice finite model capacity degrades proposal quality with increased distance to the context. Quadratic Decoding allows both steps to be done in a single efficient model call, by making K proposals of which only one is selected after verification. While this widened model call increases the computational cost, for small K the inference overhead is negligible, as the extra work is efficiently parallelized on one GPU by adopting a custom attention matrix. See Appendix H for algorithmic details.

Generally, a verification rule is set to guarantee high quality samples. In speculative decoding, based on an existing AR model, this is usually achieved by forcing the decoding algorithm to produce the same sample distribution as the reference. PTP (Draxler et al., 2026), for example, forces the generation to be token-identical to a random rollout of the AR model. Here, without a teacher to align to, this strict verification artificially harms throughput. If our model creates a plausible continuation, we avoid discarding it by loosening our acceptance rule to also admit all tokens that have high likelihood under our encoder, specifically ones that are in the ρ-nucleus of $q_\theta(x_i \mid \boldsymbol{x}_{<i})$ or ones with $q_\theta(x_i \mid \boldsymbol{x}_{<i}) \geq \epsilon$. Table 1 shows this increases the number of accepted tokens per second of the post-trained models by an average of 46.5% at identical quality.

4 Experiments

We empirically evaluate Parallel Language Models (PLMs) by their quality and wall-clock speed compared to other language model frameworks.

To this end, we pretrain PLMs of 80M, 150M, 350M, 1.3B and 3B parameters on a general mix of tokens. First, we fit scaling laws to PLMs. We find that the text quality of the encoder follows that of an identically trained autoregressive model, and that the average number of tokens the decoder can reconstruct in a single model call increases with model size, see Section 4.1.1. Compared to competing language model frameworks, PLMs achieve significant speedups Section 4.1.2. Finally, we post-train PLMs on math, coding, and instruction-following benchmarks, finding comparable or better quality compared to identically trained autoregressive models, but at about 2.3-5.7$\times$ the speed.

4.1 Pretraining

4.1.1 Scaling Laws

We study how language quality and parallelism improve with PLM size and training tokens. Quality is measured by the encoder's negative log-likelihood per token, $\mathcal{L}_{\text{enc}}$, as is standard for autoregressive models. Our parallelism metric R estimates the average number of training tokens correctly

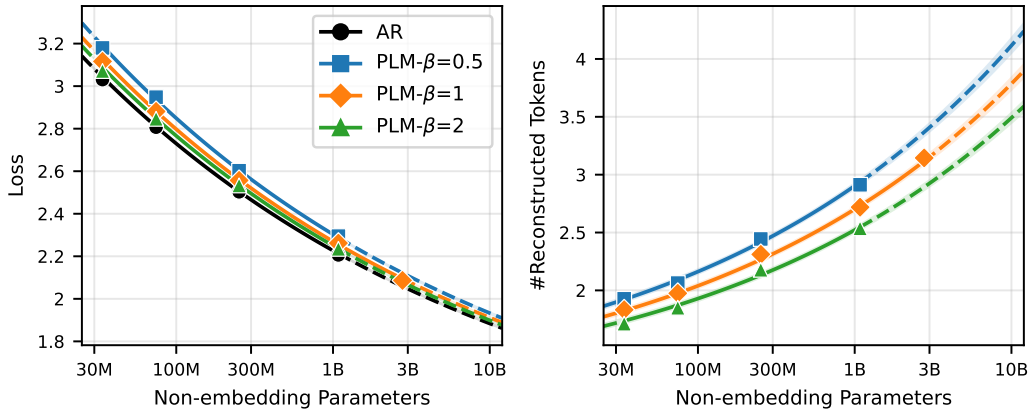

Figure 5: **PLMs become better and more parallel with scale.** *(Left)* PLMs (colored) achieve comparable quality to an AR model of the same size (black). *(Right)* PLMs achieve more correct tokens with greater model size. Varying the encoder-to-decoder ratio β reduces quality and increases speed. Markers show measurements at token to non-embedding parameter ratio $D/N = 90$, solid lines are Eq. (13).

reconstructed in parallel per model call (see Appendix G for a formal definition). An autoregressive model has $R = 1$.

We pretrain PLM-β=0.5, PLM-β=1 and PLM-β=2, corresponding to encoder-to-decoder ratios $\beta = 0.5, 1, 2$ (Section 3.2), at sizes of 80M, 150M, 350M and 1.3B parameters on ClimbMix (Diao et al., 2025). We also train matching autoregressive baselines and add a 3B PLM-β=1 extrapolation. All PLMs use decoder blocks of $K = 16$ tokens. Token budgets reach $D/N \approx 100$, where N denotes non-embedding parameters and D training tokens. Hyperparameters follow Step Law (Li et al., 2025a), with details in Appendix C.

Figure 5 shows that **larger models reconstruct more tokens at a lower loss**. We fit Chinchilla-style laws (Hoffmann et al., 2022) jointly to AR and PLM runs, modeling the loss through a normalized effective model size $n_{\text{eff}} = n\beta/(\beta + 0.48)$ and R through a power law above the autoregressive baseline:

$$\mathcal{L}_{\text{enc}} \approx \mathcal{L}_{\text{AR}}(n_{\text{eff}}, d), \quad R \approx 1 + 1.05\,\frac{n^{0.120}\,d^{0.094}}{\beta^{0.16}}, \tag{13}$$

$$\mathcal{L}_{\text{AR}}(n, d) \approx 1.105 + \frac{0.851}{n^{0.194}} + \frac{0.763}{d^{0.127}}. \tag{14}$$

Here, $n = N/10^8$ and $d = D/10^{10}$ express non-embedding parameters in units of 100M and training tokens in units of 10B. Both laws recover the autoregressive case as $\beta \to \infty$.

Over the fitted range, at fixed β, the absolute loss gap to an AR model of the same size therefore shrinks as models grow, while R increases with both model size and training tokens. Thus, as depicted in Fig. 5, **PLMs become better and more parallel with scale**, and β controls the quality–parallelism trade-off. Fit details, uncertainty estimates and the 3B extrapolation check are in Appendix G.

4.1.2 Pretraining Model Comparison

Parallel Language Models compete with other language models that allow predicting more than one token per model call. However, the latter often cannot model dependencies between tokens that are predicted in the same step, or require substantially longer training.

We therefore compare the speed and quality of publicly available autoregressive models, discrete diffusion models, and few-step discrete diffusion models to our PLMs at $D/N = 100$ from Section 4.1.1. To this end, we ask a variety of models to continue the same 200 prompts for an additional 128 tokens, where each prompt is a 64-token opening of a randomly sampled FineWeb-Edu

Table 1: **Post-trained 1.3B PLMs are comparable to or better than an autoregressive model at $2.3\times$ to $5.7\times$ the inference speed.** The 3B PLM-$\beta = 1$ model achieves better quality than all 1.3B variants. "Q / s" is questions per second. †We do not train an autoregressive model, so we use the 3B PLM in autoregressive inference mode for comparison.

Size	β	Decoding	GSM8K		MATH-500		HumanEval		IFEval	
			Score	Q / s	Score	Q / s	Score	Q / s	Score	Q / s
1.3B	AR	AR	27.9	0.9 (1.0×)	7.4	0.4 (1.0×)	12.8	0.7 (1.0×)	**48.2**	0.8 (1.0×)
	0.5	Relaxed	**32.6**	**5.2 (5.7×)**	**8.4**	**1.4 (3.7×)**	**17.7**	3.0 (4.5×)	45.1	**1.9 (2.4×)**
	1	Relaxed	29.4	4.9 (5.4×)	7.2	1.2 (3.1×)	10.4	5.2 (7.8×)	45.7	**1.9 (2.5×)**
	2	Relaxed	29.5	4.8 (5.2×)	**8.4**	1.0 (2.8×)	15.8	3.0 (4.5×)	48.1	1.8 (2.3×)
3B	1	AR†	**39.0**	0.4 (1.0×)	12.2	0.2 (1.0×)	23.2	0.4 (1.0×)	**54.0**	0.4 (1.0×)
		Relaxed	37.4	**2.1 (4.8×)**	**14.8**	**0.7 (3.8×)**	**32.3**	**1.5 (4.1×)**	53.6	**1.0 (2.5×)**

document (Lozhkov et al., 2024). Quality is measured using Phi-4 (Abdin et al., 2024) as a judge, and we fit Bradley-Terry Elo scores on the resulting data (Hunter, 2004).

Figure 1 shows that our framework **sets a new standard on the Pareto frontier of speed vs quality for pretrained models**, despite having trained on substantially fewer tokens than commercially trained models. PLMs also **outperform all tested discrete diffusion models, including few-step models**. The AR models use temperature 1 sampling among the top 128 tokens for a fair comparison to the geometry-aware auxiliaries used in our PLM model, as described in Section 3.2. We also include the actual FineWeb-Edu document completion as a human reference, which consistently scores higher than any model in the benchmark.

We benchmark PLM with strict speculative decoding, and relaxed speculative decoding as in Section 3.4. Figure 1 shows that this relaxed acceptance rule gives an increase in speed for a small drop in quality within a fixed model size. Additionally, as we increase the model size, the PLMs' ability to predict more tokens per call increases, going from 1.95 during exact speculative decoding at 350M to 2.26 at 3B.

In Appendices D and E, we report additional results and provide details on the experimental setup.

4.2 Post-training

We now confirm that Parallel Language Models can be fine-tuned to specific tasks, just like autoregressive models. To this end, we post-train our 1.3B and 3B PLMs using SFT and DPO, see Section 3.3. We also post-train our purely autoregressive 1.3B baseline.

Table 1 shows that our post-trained PLMs match or exceed the quality of an identically post-trained 1.3B autoregressive baseline, all while sampling approximately 2.3-$5.7\times$ faster, depending on the task. In addition, we confirm that our 3B PLM improves quality further, again sampling significantly faster than an autoregressive model of the same size would.

5 Conclusion

This paper derives a complete and scalable recipe to build language models from scratch that predict coordinated tokens in parallel. Already at the model scales and token budgets trained for this paper, there is a clear inference speed advantage over autoregressive models of equal quality, even over those trained with significantly higher budget.

Going forward, our mathematical framework and our scaling laws indicate no fundamental boundaries that would prevent scaling our framework to industry scales. Instead, they predict that scaled up models achieve autoregressive quality, in particular after post-training, and generate many tokens in parallel. We hope this opens a research avenue that enables autoregressive models to be as parallel in inference as they already are in training.

AI USE STATEMENT

In this work, we used generative AI tools for formulating mathematical claims, assisting in the writing of proofs design, and implementing methods. We have not used generative AI tools for helping develop theoretical models or conceptual frameworks, providing critical ingredients for proving mathematical claims, or interpreting results. The categories generating synthetic data sets, proposing or refining hypotheses, assisting with translation, cleaning and reformatting datasets, and supporting qualitative and thematic data analysis are not applicable to this work.

Additionally, we used generative AI tools for creating or modifying scientific figures or images, creating or editing software code, drafting parts of our research paper, summarizing or analyzing existing literature, sourcing/searching for information, editing our research paper to improve readability, and identifying relevant literature. We have reviewed all AI-assisted work by verifying LLM-generated code by two authors, verifying information, and by checking for potential plagiarism for research ideas through manual literature survey. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

The authors do not feel that their paper submission raises any questions regarding the ICLR Code of Ethics beyond the general risks inherent to developing and distributing AI models.

REPRODUCIBILITY STATEMENT

We have included all details necessary to replicate our experiments in Appendices C and F, and all details necessary to derive our method are in Appendix A. We will publish all our code upon publication.

REFERENCES

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

Joshua Ainslie, James Lee-Thorp, Michiel De Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 4895–4901, 2023.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. Smollm2: When smol goes big – data-centric training of a small language model, 2025. URL https://arxiv.org/abs/2502.02737.

Zachary Ankner, Rishab Parthasarathy, Aniruddha Nrusimha, Christopher Rinard, Jonathan Ragan-Kelley, and William Brandon. Hydra: Sequentially-dependent draft heads for medusa decoding. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=FbhjirzvJG.

Franz Aurenhammer. Power diagrams: properties, algorithms and applications. *SIAM journal on computing*, 16(1):78–96, 1987.

Jacob Austin, Daniel D Johnson, Jonathan Ho, Daniel Tarlow, and Rianne Van Den Berg. Structured denoising diffusion models in discrete state-spaces. *Advances in neural information processing systems*, 34:17981–17993, 2021.

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.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International conference on machine learning*, pp. 2397–2430. PMLR, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 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. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=PEpbUobfJv.

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

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

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, 2021. URL https://arxiv.org/abs/2110.14168.

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

Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Yingyan Lin, Jan Kautz, and Pavlo Molchanov. Nemotron-climb: Clustering-based iterative data mixture bootstrapping for language model pre-training, 2025. URL https://arxiv.org/abs/2504.13161.

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*, 2(3):4, 2024.

Felix Draxler, Justus Will, Farrin Marouf Sofian, Theofanis Karaletsos, Sameer Singh, and Stephan Mandt. Parallel token prediction for language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=AGJomYSrUG.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 14060–14079. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/fu24a.html.

Itai Gat, Tal Remez, Neta Shaul, Felix Kreuk, Ricky TQ Chen, Gabriel Synnaeve, Yossi Adi, and Yaron Lipman. Discrete flow matching. *Advances in Neural Information Processing Systems*, 37: 133345–133385, 2024.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, Gaël Liu, Francesco Visin, Kathleen Kenealy, Lucas Beyer, Xiaohai Zhai, Anton Tsitsulin, Robert Busa-Fekete, Alex Feng, Noveen Sachdeva, Benjamin Coleman, Yi Gao, Basil Mustafa, Iain Barr, Emilio Parisotto, David Tian, Matan Eyal, Colin Cherry, Jan-Thorsten Peter, Danila Sinopalnikov, Surya Bhupatiraju, Rishabh Agarwal, Mehran Kazemi, Dan Malkin, Ravin Kumar, David Vilar, Idan Brusilovsky, Jiaming Luo, Andreas Steiner, Abe Friesen, Abhanshu Sharma, Abheesht Sharma, Adi Mayrav Gilady, Adrian Goedeckemeyer, Alaa Saade, Alex Feng, Alexander Kolesnikov, Alexei Bendebury, Alvin Abdagic, Amit Vadi, András György, André Susano Pinto, Anil Das, Ankur Bapna, Antoine Miech, Antoine Yang, Antonia Paterson, Ashish Shenoy, Ayan Chakrabarti, Bilal Piot, Bo Wu, Bobak Shahriari, Bryce Petrini, Charlie Chen, Charline Le Lan, Christopher A. Choquette-Choo, CJ Carey, Cormac Brick, Daniel Deutsch, Danielle Eisenbud, Dee Cattle, Derek Cheng, Dimitris Paparas, Divyashree Shivakumar Sreepathihalli, Doug Reid, Dustin Tran, Dustin Zelle, Eric Noland, Erwin Huizenga, Eugene Kharitonov, Frederick Liu, Gagik Amirkhanyan, Glenn Cameron, Hadi Hashemi, Hanna Klimczak-Plucińska, Harman Singh, Harsh Mehta, Harshal Tushar Lehri, Hussein Hazimeh, Ian Ballantyne, Idan Szpektor, Ivan Nardini, Jean Pouget-Abadie, Jetha Chan, Joe Stanton, John Wieting, Jonathan Lai, Jordi Orbay, Joseph Fernandez, Josh Newlan, Ju yeong Ji, Jyotinder Singh, Kat Black, Kathy Yu, Kevin Hui, Kiran Vodrahalli, Klaus Greff, Linhai Qiu, Marcella Valentine, Marina Coelho, Marvin Ritter, Matt Hoffman, Matthew Watson, Mayank Chaturvedi, Michael Moynihan, Min Ma, Nabila Babar, Natasha Noy, Nathan Byrd, Nick Roy, Nikola Momchev, Nilay Chauhan, Noveen Sachdeva, Oskar Bunyan, Pankil Botarda, Paul Caron, Paul Kishan Rubenstein, Phil Culliton, Philipp Schmid, Pier Giuseppe Sessa, Pingmei Xu, Piotr Stanczyk, Pouya Tafti, Rakesh Shivanna, Renjie Wu, Renke Pan, Reza Rokni, Rob Willoughby, Rohith Vallu, Ryan Mullins, Sammy Jerome, Sara Smoot, Sertan Girgin, Shariq Iqbal, Shashir Reddy, Shruti Sheth, Siim Põder, Sijal Bhatnagar, Sindhu Raghuram Panyam, Sivan Eiger, Susan Zhang, Tianqi Liu, Trevor Yacovone, Tyler Liechty, Uday Kalra, Utku Evci, Vedant Misra, Vincent Roseberry, Vlad Feinberg, Vlad Kolesnikov, Woohyun Han, Woosuk Kwon, Xi Chen, Yinlam Chow, Yuvein Zhu, Zichuan Wei, Zoltan Egyed, Victor Cotruta, Minh Giang, Phoebe Kirk, Anand Rao, Kat

Black, Nabila Babar, Jessica Lo, Erica Moreira, Luiz Gustavo Martins, Omar Sanseviero, Lucas Gonzalez, Zach Gleicher, Tris Warkentin, Vahab Mirrokni, Evan Senter, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, Yossi Matias, D. Sculley, Slav Petrov, Noah Fiedel, Noam Shazeer, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Jean-Baptiste Alayrac, Rohan Anil, Dmitry, Lepikhin, Sebastian Borgeaud, Olivier Bachem, Armand Joulin, Alek Andreev, Cassidy Hardin, Robert Dadashi, and Léonard Hussenot. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Marjan Ghazvininejad, Omer Levy, Yinhan Liu, and Luke Zettlemoyer. Mask-predict: Parallel decoding of conditional masked language models. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 6112–6121, 2019.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=pEWAcejiU2.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. Non-autoregressive neural machine translation. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1l8BtlCb.

Alexander Hägele, Elie Bakouch, Atli Kosson, Loubna B Allal, Leandro Von Werra, and Martin Jaggi. Scaling laws and compute-optimal training beyond fixed training durations. *Advances in Neural Information Processing Systems*, 37:76232–76264, 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, 2021. URL https://arxiv.org/abs/2103.03874.

Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Sy2fzU9gl.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Emiel Hoogeboom, Didrik Nielsen, Priyank Jaini, Patrick Forré, and Max Welling. Argmax flows and multinomial diffusion: Learning categorical distributions. *Advances in neural information processing systems*, 34:12454–12465, 2021.

Lanxiang Hu, Siqi Kou, Yichao Fu, Samyam Rajbhandari, Tajana Rosing, Yuxiong He, Zhijie Deng, and Hao Zhang. Fast and accurate causal parallel decoding using jacobi forcing. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ORy2hyPd4G.

David R Hunter. Mm algorithms for generalized bradley-terry models. *The annals of statistics*, 32 (1):384–406, 2004.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=i1uGbfHHpH.

Chanhyuk Lee, Jaehoon Yoo, Manan Agarwal, Sheel Shah, Jerry Huang, Aditi Raghunathan, Seunghoon Hong, Nicholas M Boffi, and Jinwoo Kim. Flow map language models: One-step language modeling via continuous denoising. *arXiv preprint arXiv:2602.16813*, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Houyi Li, Wenzhen Zheng, Qiufeng Wang, Hanshan Zhang, Zili Wang, Shijie Xuyang, Yuantao Fan, Zhenyu Ding, Haoying Wang, Ning Ding, et al. Predictable scale: Part i, step law–optimal hyperparameter scaling law in large language model pretraining. *arXiv preprint arXiv:2503.04715*, 2025a.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948. PMLR, 21–27 Jul 2024a. URL https://proceedings.mlr.press/v235/li24bt.html.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 7421–7432, Miami, Florida, USA, 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.422. URL https://aclanthology.org/2024.emnlp-main.422/.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=4exx1hUffq.

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

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

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.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. Flowseq: Non-autoregressive conditional sequence generation with generative flow. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4282–4292, 2019.

Yu Meng, Mengzhou Xia, and Danqi Chen. Simpo: Simple preference optimization with a reference-free reward. *Advances in Neural Information Processing Systems*, 37:124198–124235, 2024.

Amin Karimi Monsefi, Nikhil Bhendawade, Manuel Rafael Ciosici, Dominic Culver, Yizhe Zhang, and Irina Belousova. FS-DFM: Fast and accurate long text generation with few-step diffusion language models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=ue1zFeD275.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=KnqiC0znVF.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 Furious, 2024. URL https://arxiv.org/abs/2501.00656.

Aaron Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, et al. Parallel wavenet: Fast high-fidelity speech synthesis. In *International conference on machine learning*, pp. 3918–3926. PMLR, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Krishna Pillutla, Swabha Swayamdipta, Rowan Zellers, John Thickstun, Sean Welleck, Yejin Choi, and Zaid Harchaoui. Mauve: Measuring the gap between neural text and human text using divergence frontiers. *Advances in Neural Information Processing Systems*, 34:4816–4828, 2021.

Peter Potaptchik, Jason Yim, Adhi Saravanan, Peter Holderrieth, Eric Vanden-Eijnden, and Michael S Albergo. Discrete flow maps. *arXiv preprint arXiv:2604.09784*, 2026.

Qwen Team, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=HPuSIXJaa9.

Subham S Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. *Advances in Neural Information Processing Systems*, 37:130136–130184, 2024.

Subham Sekhar Sahoo, Zhihan Yang, Yash Akhauri, Johnna Liu, Deepansha Singh, Zhoujun Cheng, Zhengzhong Liu, Eric P. Xing, John Thickstun, and Arash Vahdat. Esoteric language models: A family of any-order diffusion LLMs. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=4C5lHAIWiF.

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.

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

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

Raphael Shu, Jason Lee, Hideki Nakayama, and Kyunghyun Cho. Latent-variable non-autoregressive neural machine translation with deterministic inference using a delta posterior. In *Proceedings of the aaai conference on artificial intelligence*, number 05 in 34, pp. 8846–8853, 2020.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7b: Diffusion large language models. *arXiv preprint arXiv:2508.15487*, 2025.

Jaehoon Yoo, Wonjung Kim, Chanhyuk Lee, and Seunghoon Hong. Infinite mask diffusion for few-step distillation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=bkn1a9spwP.

Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffusion transformers is easier than you think. In *International Conference on Learning Representations*, 2025.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

A METHOD DERIVATION

In this and the following subsections, we treat encoder and decoder as arbitrary distributions with the factorizations of the main text,

$$q_\theta(\boldsymbol{x}, \boldsymbol{u}) = \prod_{i=1}^{L} q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) \prod_{i=1}^{L} q_\theta(x_i \mid \boldsymbol{x}_{<i}), \tag{15}$$

$$p_\theta(\boldsymbol{x}, \boldsymbol{u}) = \prod_{i=1}^{L} \underbrace{p_\theta(x_i \mid \boldsymbol{u}_{\leq i})}_{>0} \prod_{i} \underbrace{p(u_i)}_{>0}. \tag{16}$$

This is justified because we only consider model architectures with these properties.

A.1 EVIDENCE LOWER BOUND

We prove Eq. (4), i.e. that

$$\mathcal{L}(\theta; \boldsymbol{x}) \coloneqq \underbrace{\mathbb{E}_{\boldsymbol{u} \sim q_\theta(\boldsymbol{u} \mid \boldsymbol{x})} \left[\sum_{i=1}^{L} -\log p_\theta(x_i \mid \boldsymbol{u}_{\leq i}) \right]}_{\mathcal{L}_{\mathrm{dec}}(\theta; \boldsymbol{x})} + \sum_{i=1}^{L} D_{\mathrm{KL}}(q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) \parallel p(u_i)) \tag{17}$$

is an upper bound to the negative likelihood. To this end, we show

$$-\log p_\theta(\boldsymbol{x}) = \mathcal{L}(\theta; \boldsymbol{x}) - D_{\mathrm{KL}}(q_\theta(\boldsymbol{u} \mid \boldsymbol{x}) \parallel p_\theta(\boldsymbol{u} \mid \boldsymbol{x})), \tag{18}$$

where

$$p_\theta(\boldsymbol{u} \mid \boldsymbol{x}) = \frac{\prod_{i=1}^{L} p(u_i) \prod_{i=1}^{L} p_\theta(x_i \mid \boldsymbol{u}_{\leq i})}{p_\theta(\boldsymbol{x})} \tag{19}$$

denotes the true posterior over auxiliaries. The result follows as D_{KL} is non-negative. From our assumptions follows $\mathrm{supp}(q_\theta(\boldsymbol{u} \mid \boldsymbol{x})) \subseteq \mathrm{supp}(p_\theta(\boldsymbol{u} \mid \boldsymbol{x}))$, and since $-\log p_\theta(\boldsymbol{x})$ does not depend on $\boldsymbol{u}$, we can take its expectation under $q_\theta(\boldsymbol{u} \mid \boldsymbol{x})$ and apply Bayes' rule:

$$-\log p_\theta(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{u} \sim q_\theta(\boldsymbol{u} \mid \boldsymbol{x})} \left[-\log \frac{\prod_{i=1}^{L} p(u_i) \prod_{i=1}^{L} p_\theta(x_i \mid \boldsymbol{u}_{\leq i})}{p_\theta(\boldsymbol{u} \mid \boldsymbol{x})} \right] \tag{20}$$

$$= \mathbb{E}_{\boldsymbol{u} \sim q_\theta(\boldsymbol{u} \mid \boldsymbol{x})} \left[-\log \left(\prod_{i=1}^{L} p_\theta(x_i \mid \boldsymbol{u}_{\leq i}) \cdot \frac{\prod_{i=1}^{L} p(u_i)}{\prod_{i=1}^{L} q_\theta(u_i \mid \boldsymbol{x}_{\leq i})} \cdot \frac{q_\theta(\boldsymbol{u} \mid \boldsymbol{x})}{p_\theta(\boldsymbol{u} \mid \boldsymbol{x})} \right) \right] \tag{21}$$

$$= \mathcal{L}_{\mathrm{dec}}(\theta; \boldsymbol{x}) + \sum_{i=1}^{L} D_{\mathrm{KL}}(q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) \parallel p(u_i)) - D_{\mathrm{KL}}(q_\theta(\boldsymbol{u} \mid \boldsymbol{x}) \parallel p_\theta(\boldsymbol{u} \mid \boldsymbol{x})), \tag{22}$$

where the second equality multiplies and divides by $q_\theta(\boldsymbol{u} \mid \boldsymbol{x})$ inside the logarithm, the third splits the logarithm and uses linearity of expectation. This shows Eq. (18), and Eq. (4) follows.

A.2 TIGHTNESS OF THE BOUND AT THE GLOBAL MINIMUM

During training, we optimize $\mathcal{L}(\theta) = \mathbb{E}_{\boldsymbol{x} \sim p_{\mathrm{data}}}[\mathcal{L}(\theta; \boldsymbol{x})]$. Averaging Eq. (18) gives

$$\mathcal{L}(\theta) = \underbrace{\mathbb{E}_{\boldsymbol{x} \sim p_{\mathrm{data}}}[-\log p_\theta(\boldsymbol{x})]}_{\geq H(p_{\mathrm{data}})} + \underbrace{\mathbb{E}_{\boldsymbol{x} \sim p_{\mathrm{data}}}[D_{\mathrm{KL}}(q_\theta(\boldsymbol{u} \mid \boldsymbol{x}) \parallel p_\theta(\boldsymbol{u} \mid \boldsymbol{x}))]}_{\geq 0} \geq H(p_{\mathrm{data}}). \tag{23}$$

The bound is tight at the global minimum, that is attained if and only if $p_\theta = p_{\mathrm{data}}$ and $q_\theta(\boldsymbol{u} \mid \boldsymbol{x}) = p_\theta(\boldsymbol{u} \mid \boldsymbol{x})$ almost everywhere. Then, the cross-entropy $\mathbb{E}_{\boldsymbol{x} \sim p_{\mathrm{data}}}[-\log p_\theta(\boldsymbol{x})]$ equals the data entropy $H(p_{\mathrm{data}})$, and the D_{KL} term vanishes.

A.3 Encoder Loss vs Autoregressive loss

In the following we will restrict our encoder to be injective, i.e. such that for fixed $\boldsymbol{x}_{<i}$ and all $x_i \neq x_i'$ their supports $\mathcal{C}_{x_i} := \operatorname{supp} q_\theta(u_i \mid \boldsymbol{x}_{<i}, x_i)$ and $\mathcal{C}_{x_i'}$ are disjoint. Appendix A.4 shows that we can construct such an encoder that still is a global minimizer of Eq. (23).

Theorem 1. *Then, for any injective encoder q_θ, for all x_i,*

$$D_{\mathrm{KL}}\big(q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) \parallel p(u_i)\big) = -\log q_\theta(x_i \mid \boldsymbol{x}_{<i}) \tag{24}$$

if and only if $q_\theta(u_i \mid \boldsymbol{x}_{<i}) = p(u_i)$ almost everywhere.

Proof. By disjointness, for $u_i \in \mathcal{C}_{x_i}$ only one term survives in the aggregate posterior:

$$q_\theta(u_i \mid \boldsymbol{x}_{<i}) = \sum_{x_i} q_\theta(u_i \mid \boldsymbol{x}_{<i}, x_i)\, q_\theta(x_i \mid \boldsymbol{x}_{<i}) = q_\theta(u_i \mid \boldsymbol{x}_{\leq i})\, q_\theta(x_i \mid \boldsymbol{x}_{<i}). \tag{25}$$

For such u_i, $q_\theta(x_i \mid u_i, \boldsymbol{x}_{<i}) = 1$ by construction and Bayes' rule yields

$$q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) = q_\theta(u_i \mid \boldsymbol{x}_{<i}) / q_\theta(x_i \mid \boldsymbol{x}_{<i}). \tag{26}$$

Substituting into the KL gives

$$D_{\mathrm{KL}}\big(q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) \parallel p(u_i)\big) = -\log q_\theta(x_i \mid \boldsymbol{x}_{<i}) + \mathbb{E}_{q_\theta(u_i \mid \boldsymbol{x}_{\leq i})}\left[\log \frac{q_\theta(u_i \mid \boldsymbol{x}_{<i})}{p(u_i)}\right]. \tag{27}$$

($\Leftarrow$) If $q_\theta(u_i \mid \boldsymbol{x}_{<i}) = p(u_i)$ a.e., the last term of Eq. (27) vanishes.

($\Rightarrow$) If the equality holds for all x_i, the last term of Eq. (27) is zero for every x_i. Averaging over $x_i \sim q_\theta(x_i \mid \boldsymbol{x}_{<i})$ and using $\sum_{x_i} q_\theta(x_i \mid \boldsymbol{x}_{<i})\, q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) = q_\theta(u_i \mid \boldsymbol{x}_{<i})$ yields

$$0 = \mathbb{E}_{q_\theta(u_i \mid \boldsymbol{x}_{<i})}\left[\log \frac{q_\theta(u_i \mid \boldsymbol{x}_{<i})}{p(u_i)}\right] = D_{\mathrm{KL}}\big(q_\theta(u_i \mid \boldsymbol{x}_{<i}) \parallel p(u_i)\big),$$

so $q_\theta(u_i \mid \boldsymbol{x}_{<i}) = p(u_i)$ a.e. □

The condition $q_\theta(u_i \mid \boldsymbol{x}_{<i}) = p(u_i)$ enforces that u_i only contains the additional information about x_i. We can ensure this by construction:

Proposition 1. *For any injective encoder $q_\theta(u_i \mid \boldsymbol{x}_{<i}) = p(u_i)$ if and only if*

$$q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) = \frac{p(u_i)\, \mathbf{1}\{u_i \in \mathcal{C}_{x_i}\}}{q_\theta(x_i \mid \boldsymbol{x}_{<i})},$$

i.e. the ratio $q_\theta(u_i \mid \boldsymbol{x}_{\leq i})/p(u_i)$ is constant on $\mathcal{C}_{x_i}$ (shape matching) and $p(\mathcal{C}_{x_i}) = q_\theta(x_i \mid \boldsymbol{x}_{<i})$ (mass matching).

Proof. ($\Rightarrow$) By disjointness on $\mathcal{C}_{x_i}$, and substituting $q_\theta(u_i \mid \boldsymbol{x}_{<i}) = p(u_i)$ into Eq. (26).

($\Leftarrow$) Integrating over $\mathcal{C}_{x_i}$ gives $p(\mathcal{C}_{x_i}) = q_\theta(x_i \mid \boldsymbol{x}_{<i})$, so $\sum_{x_i} p(\mathcal{C}_{x_i}) = 1$. Hence,

$$q_\theta(u_i \mid \boldsymbol{x}_{<i}) = \sum_{x_i} q_\theta(x_i \mid \boldsymbol{x}_{<i})\, \frac{p(u_i)\, \mathbf{1}\{u_i \in \mathcal{C}_{x_i}\}}{q_\theta(x_i \mid \boldsymbol{x}_{<i})} = p(u_i).$$ □

Corollary 1 (Voronoi auxiliaries)**.** *An encoder that uses the Voronoi embedding defined in Section 3.2 fulfills the equality in Eq.* (24)*.*

Proof. We directly apply Theorem 1 and Proposition 1.

Injectivity: By construction, the Voronoi cells $\mathcal{C}_v$ of Eq. (8) are disjoint for all $v = x_i$.

Shape matching: The prior $p(u) = \mathrm{Unif}(\mathcal{U})$, is uniform over the discrete auxiliary space. As we draw embeddings randomly over their cell, we have $q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) = \mathrm{Unif}(\mathcal{C}_{x_i})$, and constant ratio.

Mass matching: The fitted offsets in Eq. (9) are optimized to yield $p(\mathcal{C}_{x_i}) = q_\theta(x_i \mid \boldsymbol{x}_{<i})$ up to rounding errors that vanish as $A \to \infty$. □

A.4 Encoder Injectivity at Optimality

In the following, we relax our assumptions to $p_\theta(x_i \mid \boldsymbol{u}_{\leq i}) \geq 0$, i.e. the decoder is allowed to vanish. This is not a restriction, as any such decoder can be approximated arbitrarily close by a decoder with strict positivity.

For a fixed injective encoder, $\boldsymbol{u}_{\leq i}$ determines $\boldsymbol{x}_{\leq i}$, say $x_i = g_i(\boldsymbol{u}_{\leq i})$. Then, $\mathcal{L}_{\mathrm{dec}}$ is a sum of non-negative terms $-\log p_\theta(x_i \mid \boldsymbol{u}_{\leq i})$ that vanishes if and only if $p_\theta(x_i \mid \boldsymbol{u}_{\leq i}) = \mathbf{1}[x_i = g_i(\boldsymbol{u}_{\leq i})]$: the optimal decoder is the deterministic inverse of the encoder. Inserting it into Eq. (3),

$$p_\theta(\boldsymbol{x}) = \int \prod_{i=1}^{L} p(u_i)\, \mathbf{1}[u_i \in \mathcal{C}_{x_i}]\, d\boldsymbol{u} = \prod_{i=1}^{L} p(\mathcal{C}_{x_i}), \qquad \text{so} \qquad p_\theta(x_i \mid \boldsymbol{x}_{<i}) = p(\mathcal{C}_{x_i}) \tag{28}$$

when the cells cover the auxiliary space $\mathcal{U}$: the probability of a token is the prior mass of its cell.

If the injective encoder additionally satisfies $p(\mathcal{C}_{x_i}) = p_{\mathrm{data}}(x_i \mid \boldsymbol{x}_{<i})$ and $q_\theta(u_i \mid \boldsymbol{x}_{\leq i}) = p(u_i)\mathbf{1}\{u_i \in \mathcal{C}_{x_i}\}/p(\mathcal{C}_{x_i})$, we have $p_\theta = p_{\mathrm{data}}$ and $q_\theta(\boldsymbol{u} \mid \boldsymbol{x}) = p_\theta(\boldsymbol{u} \mid \boldsymbol{x})$ almost everywhere, which yields global optimality as in Appendix A.2, showing that injectivity is not a constraint for optimality.

B Fitting the Voronoi Weights and Alternative Cell Geometries

At position i, let T_i be the M most likely tokens under the encoder distribution. Their renormalized probabilities are

$$\tilde{q}_{i,v} = \frac{q_\theta(x_i = v \mid \boldsymbol{x}_{<i})}{\sum_{v' \in T_i} q_\theta(x_i = v' \mid \boldsymbol{x}_{<i})}, \qquad v \in T_i. \tag{29}$$

We convert these probabilities into integer cell capacities $n_{i,v}$ by largest-remainder rounding: each token first receives $\lfloor A\tilde{q}_{i,v} \rfloor$ auxiliaries, and the remaining auxiliaries are allocated in descending order of the fractional remainders. The capacities therefore satisfy $\sum_{v \in T_i} n_{i,v} = A$.

Laguerre cells. We use the unnormalized output-layer rows $\boldsymbol{e}_v$ as cell anchors and fixed unit vectors $\boldsymbol{e}_u \in \mathbb{S}^{d-1}$ as auxiliary embeddings. The score $s_{i,v}(u) = \boldsymbol{e}_v \cdot \boldsymbol{e}_u + w_{i,v}$ defines the cells $\mathcal{C}_{i,v}$ through Eq. (8). This gives a Laguerre diagram with sites $\boldsymbol{e}_v$ and power weights $\|\boldsymbol{e}_v\|_2^2 + 2w_{i,v}$, restricted to the auxiliary sphere:

$$\|\boldsymbol{e}_u - \boldsymbol{e}_v\|_2^2 - \big(\|\boldsymbol{e}_v\|_2^2 + 2w_{i,v}\big) = 1 - 2s_{i,v}(u). \tag{30}$$

Thus maximizing the score is equivalent to minimizing the weighted squared distance. The offsets $w_{i,v}$ control the cell sizes while retaining the dot-product geometry of the language-model head.

Weight fitting. The offsets are dual variables of the balanced assignment problem that maximizes the total dot-product score subject to the capacities $n_{i,v}$. We fit them using annealed Sinkhorn updates. At temperature $\tau > 0$, the soft assignments and corresponding cell loads are

$$\pi^\tau_{i,v,u} = \frac{\exp\big(s_{i,v}(u)/\tau\big)}{\sum_{v' \in T_i} \exp\big(s_{i,v'}(u)/\tau\big)}, \qquad \ell^\tau_{i,v} = \sum_{u \in \mathcal{U}} \pi^\tau_{i,v,u}. \tag{31}$$

For $n_{i,v} > 0$, the log-domain update is

$$w_{i,v} \leftarrow w_{i,v} + \tau\big(\log n_{i,v} - \log \ell^\tau_{i,v}\big). \tag{32}$$

Table 2: **Model sizes and per-run settings.** N counts non-embedding parameters. "Total" adds the untied input and output embeddings. The planned token budget $D_{\max}$ sets the learning rate η and the global batch size (GBS, in sequences of 4096 tokens) through Eq. (33). †The 3B constant-learning-rate run stopped at 240B tokens; its η and GBS are those for the planned 300B.

Model	Layers	d	FFN	Heads	KV	N	Total	$D_{\max}$	GBS	η	Nodes
80M	12	512	1408	8	2	33.8M	85.3M	3B	32	$6.22 \cdot 10^{-3}$	1
150M	12	768	2048	12	3	74.3M	151.6M	6B	64	$4.39 \cdot 10^{-3}$	2
350M	24	1024	2560	16	4	251.7M	354.7M	20B	128	$2.66 \cdot 10^{-3}$	4
1.3B	24	2048	5632	32	8	1.08B	1.29B	80B	256	$1.44 \cdot 10^{-3}$	8
3B	28	3072	8192	48	12	2.77B	3.08B	300B†	512	$1.10 \cdot 10^{-3}$	32

This increases the scores of underfilled cells and decreases those of overfilled cells. Candidates with zero capacity receive a large negative offset. We initialize the offsets from the capacity shares and the scale of the geometric scores, then apply clipped, over-relaxed updates as τ decreases. To reduce the fitting cost, we evaluate the soft assignments over a small shortlist of candidates for each auxiliary and refresh these shortlists between rounds. The final assignment takes the highest score over all M candidates, with consistent tie-breaking. We use a fixed number of updates, so the resulting cell counts approximate the prescribed capacities; the fitting settings and measured error are reported in Appendix C.

C Pretraining Details

This section lists the settings of the pretraining runs in Section 4.1.1. PLMs and autoregressive (AR) baselines of the same size share every setting except the loss. The AR baselines switch off the decoder and alignment losses.

Data. We pretrain on ClimbMix (Diao et al., 2025), tokenized with the GPT-2 byte-pair encoding (Radford et al., 2019). Its $V = 50{,}257$ tokens are padded to $50{,}304$ embedding rows, a multiple of 128. The tokenized corpus holds 357.4B tokens in 553M documents. The largest run, the 3B PLM, reads 300B tokens (240B constant learning rate and 60B during learning rate decay), or 84% of the corpus. We hold out no split and report training losses throughout.

Training sequences are 4096 tokens long and packed across document boundaries, with an end-of-document token between documents. The AR loss covers every position of a packed sequence. A completion block, in contrast, ends at the first end-of-document token among its targets, so it never reconstructs tokens from two documents.

Architecture. All models are decoder-only transformers. They use pre-normalization with RMSNorm (Zhang & Sennrich, 2019), SwiGLU MLPs (Shazeer, 2020), and grouped-query attention (Ainslie et al., 2023). In grouped-query attention, four query heads share each key–value head, and every head has dimension 64. Rotary embeddings (Su et al., 2024) with base $500{,}000$ encode the positions. The linear layers of the transformer have no biases, the input and output embeddings are untied, and we use no dropout. All weights are initialized from $\mathcal{N}(0, 0.02^2)$. Table 2 lists the five model sizes.

The parameter count N of the scaling laws counts only the transformer body. It excludes the input and output embeddings and the few parameters the PLM adds: the alignment map g_ψ and the mask auxiliary described below.

Optimizer. We use AdamW (Loshchilov & Hutter, 2017) with $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\epsilon = 10^{-8}$, and weight decay 0.1. Gradients are clipped to a global norm of 1. We train in bfloat16 mixed precision and keep gradient reductions, master weights, and optimizer states in fp32.

Learning rate and batch size. We set the peak learning rate and the global batch size from N and the token budget $D_{\max}$ of the constant-learning-rate run with Step Law (Li et al., 2025a), a fitted

rule for these two hyperparameters:

$$\eta(N, D_{\max}) = 1.79\, N^{-0.713} D_{\max}^{0.307}, \qquad B(D_{\max}) = 0.58\, D_{\max}^{0.571} \text{ tokens.} \tag{33}$$

We round B to the nearest power of two in sequences. The micro-batch size trades memory for throughput and does not change the optimization.

For the 3B model, both values come from the planned budget $D_{\max} = 300$B, but we stopped its constant-learning-rate run at 240B tokens (see below). At 240B, Eq. (33) gives $\eta = 1.03 \cdot 10^{-3}$, 7% below the value we used, and the same GBS of 512 after rounding. The 3B hyperparameters are therefore close to, but not exactly, those Step Law prescribes for the run we trained.

Learning rate schedule. Each configuration trains one long run with a warmup–stable–decay schedule (Hägele et al., 2024). The learning rate warms up linearly over 500 iterations and then stays at η until the run reaches its budget $D_{\max}$. We keep about ten evenly spaced checkpoints along this run.

From each checkpoint, an annealing branch decays the learning rate on the unseen part of the data stream. The branch from iteration t trains for another $T = t/4$ iterations. It decays the learning rate from η to $\eta_{\min} = 10^{-5}$ with the square-root cooldown of Hägele et al. (2024):

$$\eta_\tau = \eta_{\min} + (\eta - \eta_{\min})\big(1 - \sqrt{\tau/T}\big), \qquad \tau = 0, \ldots, T. \tag{34}$$

The branch thus trains for $5t/4$ iterations in total, and its token count is the D it contributes to the fits of Appendix G.

One constant-learning-rate run per configuration therefore yields about ten fully decayed models with different D. From 80M to 1.3B, the last branch starts where the constant-learning-rate run ends, so it ends at $5/4$ of the budget; the 1.3B models, for example, end at iteration $95{,}366$ (100B tokens). The 3B constant-learning-rate run stopped earlier, at iteration $114{,}436$ (240B tokens, $4/5$ of its budget $D_{\max} = 300$B). Its last branch therefore ends exactly at the budget, at iteration $143{,}045$ (300B tokens). The post-trained models of Table 11 start from these final branches.

Blocks and loss ratio. All PLMs use block size $K = 16$. The stride s of Eq. (7) sets $\beta = s/K$, the number of encoder positions per decoder position, which weights the encoder loss against the decoder loss like the β of a β-VAE (Higgins et al., 2017). PLM-β=0.5, PLM-β=1, and PLM-β=2 use $s = 8$, 16, and 32.

The block grid shifts at every step, so every position starts a block equally often over training. Each step draws a phase o uniformly from $\{0, \ldots, s-1\}$, independently on each data-parallel rank. Block b then starts at $i_b = (b-1)s+1-o$. Blocks that extend past either end of the sequence are truncated.

Loss normalization. Every scored position enters the loss with unit weight. We divide Eq. (7) by the number of scored positions: every encoder position of the sequence, plus about $1/\beta$ decoder positions per encoder position. The ratio β of encoder to decoder positions is therefore also the weight of the encoder loss relative to the decoder loss. We add the alignment loss Eq. (10), weighted by λ, to the numerator before the division, so it is normalized in the same way as the two cross-entropy terms.

Training step. Each step runs the encoder and the decoder as two FlexAttention (Dong et al., 2024) passes. The encoder pass caches its keys and values, and the decoder pass attends to this cache instead of recomputing the clean prefix. The cache stays in the computation graph, so $\mathcal{L}_{\text{dec}}$ also backpropagates into the encoder pass.

Auxiliaries. We use $A = 512$ auxiliaries and $M = 128$ candidate tokens. The quotas of Eq. (9) are rounded by largest remainder, so they sum to exactly A. The offset fit of Appendix B runs 2 rounds of 12 sweeps over a shortlist of 8 candidates per auxiliary. It leaves about 1% of auxiliaries outside their quota. Some observed tokens have no auxiliary, because they fall outside the top M or receive a quota of zero. At these positions, the decoder reads a learned mask embedding instead.

Table 3: **GPU-hours of the final pretrained models**, including the final annealing branch. The 3B model was trained only as PLM-β=1; its constant-learning-rate run moved from 64 to 128 GPUs after 31 hours.

Model	GPUs	AR	PLM-β=2 ($s = 32$)	PLM-β=1 ($s = 16$)	PLM-β=0.5 ($s = 8$)
350M	16	98	172	201	285
1.3B	32	761	1,436	1,884	2,701
3B	64–128	–	–	11,872	–

Representation alignment. We use $\lambda = 5$, $\ell_{\text{enc}} = \lfloor \text{Layers}/2 \rfloor$, and $\ell_{\text{dec}} = \lfloor \text{Layers}/3 \rfloor$. This gives layers $(\ell_{\text{enc}}, \ell_{\text{dec}}) = (6, 4)$ at 80M and 150M, $(12, 8)$ at 350M and 1.3B, and $(14, 9)$ at 3B. Both states are read from the residual stream between the attention and MLP blocks of their layer. The map g_ψ is a linear layer with bias.

Infrastructure. We implement PLMs in PyTorch (Paszke et al., 2019) on top of Megatron-LM (Shoeybi et al., 2019) and NVIDIA Transformer Engine. All models train on NVIDIA GH200 nodes with four GPUs each, using the node counts in Table 2. Training uses only data parallelism, without tensor, pipeline, or context parallelism, and shards optimizer states across data-parallel ranks.

Compute. Table 3 lists the GPU-hours of the final model of each size and objective. Each figure sums the wall-clock time of every job that trained the model, times the GPUs of that job. It covers the constant-learning-rate run up to its last branch point plus the final annealing branch. We leave out the 80M and 150M models, whose wall-clock time is dominated by stalls in which a step waits on data loading or the file system, rather than by computation.

PLMs cost more than AR baselines because each step processes $1 + 1/\beta$ times the positions of an AR step: $1.5\times$ for PLM-β=2, $2\times$ for PLM-β=1, and $3\times$ for PLM-β=0.5. Relative to the AR baseline of the same size, PLM-β=2 costs $1.89\times$ at 1.3B, PLM-β=1 costs $2.48\times$, and PLM-β=0.5 costs $3.55\times$, due to additional overheads caused by Voronoi auxiliaries. Between PLMs, going from PLM-β=2 to PLM-β=1 multiplies the positions by 1.33 and the GPU-hours by 1.17–1.31. Going on to PLM-β=0.5 multiplies the positions by 1.5 and the GPU-hours by 1.42–1.43.

D Ablation on Design Choices

We ablate the design choices of Section 3.2 on a small scale. The models use the same data and schedule as the scaling runs of Section 4.1.1. Unless stated otherwise, we use the 150M model (see Appendix C).

We report three metrics, each averaged over the last 200 training steps of a run. The *AR NLL* is the loss of the autoregressive encoder, $-\log q_\theta(x_i \mid \boldsymbol{x}_{<i})$, on the clean tokens. It measures model quality and is directly comparable to the loss of an AR model. The *reconstructed tokens* R count how many tokens the decoder predicts correctly per block. The encoder draws auxiliaries for the true continuation of a block, and R is the mean length of the block prefix that the decoder then reconstructs exactly by argmax. The first position of a block always counts as reconstructed, so an AR model has $R = 1$.

Every data point comes from a single run with a single seed. As a reference for the noise level, extending a finished learning rate decay training by 20% more steps at a lower learning rate moves the AR NLL by at most 0.008 nats and R by at most 0.004.

D.1 Auxiliary Choice

We show in Table 4 how the choice of auxiliaries changes the model performance.

We start from a model that uses auxiliary positions which carry no information. Every auxiliary position holds the same single token, so the decoder must guess the continuation from the clean prefix alone, as a multi-token prediction head does.

Auxiliaries	Alignment	AR NLL	R
AR model		2.788	1
None	no	2.821	1.443
None	yes	2.807	1.464
Quantile	no	2.849	1.705
Quantile	yes	2.837	1.739
Voronoi	no	2.844	1.785
Voronoi	yes	2.829	1.850

Table 4: **Building the parallel decoder up from an AR model.** 150M models at stride $s = 32$ ($\beta = 2$), trained for 7.5B tokens including the learning rate decay. The last row is our recipe. The AR model uses the same schedule.

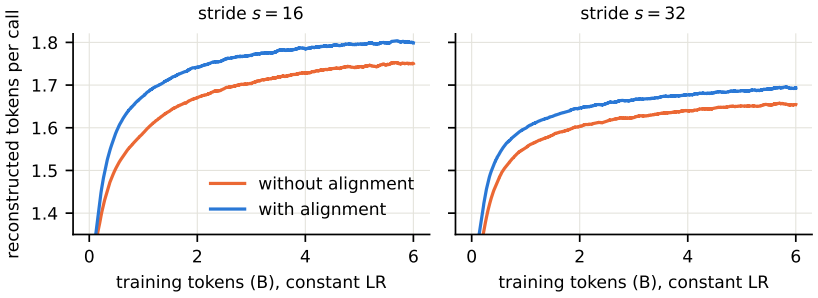

Figure 6: **Alignment reaches the same number of reconstructed tokens in fewer steps.** Reconstructed tokens per model call R of the 150M PLM (block size $K = 16$, Voronoi auxiliaries) with alignment (blue) and without (orange). Each curve is one training run during its constant learning rate phase of 6B tokens, before learning rate decay. *Left:* stride $s = 16$ ($\beta = 1$). *Right:* stride $s = 32$ ($\beta = 2$).

Quantile auxiliaries follow Draxler et al. (2026) and are the next step. They cut the unit interval into 512 buckets, and each bucket is one auxiliary token. We sort the vocabulary by token index and give each token the interval of quantiles it covers under $q_\theta(x_i \mid \boldsymbol{x}_{<i})$. A token then owns the buckets that fall into its interval. Quantile auxiliaries without alignment are the closest variant to the PTP recipe of Draxler et al. (2026). Quantile auxiliaries raise R from 1.443 without (meaningful) auxiliaries to 1.705.

Voronoi auxiliaries further raise R over quantile auxiliaries, from 1.705 to 1.785 without representation alignment.

D.2 Representation alignment

We show how representation alignment changes the performance of the parallel decoder with all three auxiliary settings in Table 4.

Representation alignment improves the final scores in every setting, raising the final number of reconstructed tokens from 1.785 to 1.850 and lowering the final AR NLL from 2.844 to 2.829.

Without alignment, the model needs 1.7 to 2.7 times the training steps for the same number of reconstructed tokens. Figure 6 compares our recipe with and without the alignment loss of Eq. (10) during the constant learning rate phase of 6B tokens. The aligned model reaches the R that the unaligned model has at the end of this phase after 37% of the steps at stride 16 and after 38% at stride 32. The saving is smaller earlier in training. To reach the R that the unaligned model has at 1B tokens, the aligned model needs 49% of the steps at stride 16 and 58% at stride 32.

Alignment speeds up the AR NLL less than it speeds up R. To match the final AR NLL of the unaligned model, the aligned model needs 53% to 59% of the steps at constant learning rate. After learning rate decay at stride 32, it needs 89% of the training tokens.

D.3 RANGE OF STRIDES

Smaller strides train more auxiliary positions per clean token. The stride s sets this number to K/s, so a training step processes $1 + K/s$ times the tokens of an AR step. The main text uses $s \in \{8, 16, 32\}$, and Table 5 extends the sweep to $s \in \{2, 4, 64\}$ with 80M models.

Strides below 8 buy few reconstructed tokens at a large cost in quality and compute. Going from stride 8 to stride 2 triples the tokens per step and adds only 0.11 reconstructed tokens. It also worsens the AR NLL by 0.24 nats, more than the whole gap between stride 8 and the AR model.

Stride 64 leaves little room for a speedup. Its AR NLL is within 0.02 nats of the AR model, but it reconstructs only 0.6 extra tokens per call.

Stride s	β	Tokens per step	AR NLL	R
AR	–	$1\times$	3.023	1
64	4	$1.25\times$	3.041	1.598
32	2	$1.5\times$	3.065	1.719
16	1	$2\times$	3.110	1.844
8	0.5	$3\times$	3.174	1.936
4	0.25	$5\times$	3.266	1.987
2	0.125	$9\times$	3.410	2.046

Table 5: **Reconstructed tokens saturate at small strides while quality keeps degrading.** 80M models trained for 3.75B tokens including the learning rate decay. "Tokens per step" is the number of positions processed per training step relative to an AR model, $1 + K/s$ at $K = 16$.

D.4 TRAINING FROM SCRATCH VERSUS CONVERTING AN AR MODEL

Converting a trained AR model is a cheaper alternative to training a PLM from scratch. We train an AR model and switch to the PLM objective (Eq. (7) with alignment) only for the learning rate decay, the final 20% of training. The conversion thus saves the decoder compute of the first 80% of training. We branch the conversions off the AR runs at the same steps and with the same learning rate decay as the PLM runs. Both therefore see the same data and compute during the decay.

The converted model reconstructs fewer tokens than the PLM trained from scratch (Table 6). At every size and stride, it makes 0.15 to 0.28 fewer reconstructed tokens per call.

The conversion does not improve model quality either. Its AR NLL is worse by 0.01 to 0.025 nats at strides 16 and 32 and within 0.01 nats at stride 8. Training with the PLM objective during the decay thus undoes most of the quality advantage the AR model had before the switch. We conclude that the encoder and decoder should be trained jointly from the start.

			AR NLL			Reconstructed tokens R	
Size	Tokens	Stride	from scratch	converted	AR model	from scratch	converted
80M	3.75B	8	3.174	3.181	3.023	1.936	1.734
80M	3.75B	16	3.110	3.124	3.023	1.844	1.649
80M	3.75B	32	3.065	3.085	3.023	1.719	1.566
150M	7.5B	8	2.930	2.926	2.788	2.081	1.835
150M	7.5B	16	2.863	2.878	2.788	1.993	1.747
150M	7.5B	32	2.829	2.844	2.788	1.850	1.647
350M	25B	32	2.531	2.555	2.500	2.182	1.919
1.3B	100B	32	2.235	2.255	2.206	2.537	2.260

Table 6: **Converting an AR model is worse than training from scratch.** "Converted" trains an AR model at constant learning rate and switches to the PLM objective for the learning rate decay, the final 20% of training. "AR model" applies the same decay without the switch. Token counts include the decay.

E PRETRAINING EVALUATION DETAILS

This appendix describes how we compare the generation quality and speed of pretrained PLMs with autoregressive, diffusion and flow baselines (Section 4.1.2), and reports the measurements behind that comparison. Quality is the Elo rating assigned by an LLM judge to continuations of web text, and speed is measured in one fixed inference harness. Implementation details of the harness and of each baseline family are in Appendices H.1 to H.5.

E.1 EVALUATION PROMPTS

We draw prompts from FineWeb-Edu (Lozhkov et al., 2024), snapshot CC-MAIN-2025-26, crawled in June 2025. This is after the release of ClimbMix (Diao et al., 2025) in February 2025, which was built from crawls up to 2024 and is our pretraining corpus. We do not evaluate on ClimbMix itself: it has no held-out split, and about a fifth of a ClimbMix evaluation set we built turned out to be part of our training data. Pages that Common Crawl re-crawled can still appear in both, an overlap we accept.

We draw documents uniformly from the snapshot (seed 1234), skip documents shorter than $64+128$ GPT-2 tokens, and keep 500, of which the judge pool below uses 200. A prompt is the first 64 GPT-2 tokens of a document, i.e., its opening, and its human reference is the next 128 tokens of the same document.

E.2 GENERATION

Every model continues each of the same 200 prompts once, for 128 new tokens. We continue generating after end-of-tokens. The human reference takes part in the comparison as a player of its own.

Sampling. Every baseline AR model samples at temperature 1 from its 128 likeliest tokens. A PLM decodes with its Voronoi sampler (Section 3.2), which is truncated by construction, because only the $M = 128$ likeliest tokens receive auxiliaries. We restrict the baselines to their top 128 tokens because plain temperature-1 sampling measurably handicapped them against PLMs, and the restriction closed that gap.

E.3 DECODING MODES AND SPEED

Harness. All autoregressive models, all PLMs, LLaDA and Dream are served by vLLM (Kwon et al., 2023) on a single NVIDIA RTX 6000 Ada, one prompt at a time (batch size 1). Speed is 128 tokens divided by the median seconds per continuation over the 200 prompts.

Our models. All our rows come from one decoder build for every size and stride (float16, round width $k = 6$). Each PLM checkpoint is decoded in three ways:

- **AR sampling:** one token per model call with a top-128 cut, as for the baselines;
- **exact:** lossless speculative decoding, which accepts a drafted token only if it matches the model's own draw exactly;
- **relaxed:** lossy verification, which accepts a drafted token if the model assigns it a probability above $2 \cdot 10^{-3}$.

We chose the relaxed rule as the fastest rule whose MAUVE (Pillutla et al., 2021) stays within 1% of exact decoding, measured on 500 FineWeb-Edu prompts with 128 generated tokens each and averaged over six MAUVE scorer seeds. Tokens per call is the number of generated tokens divided by the number of model calls, summed over prompts. It is 1 for every autoregressive decoding.

The main comparison shows stride 16 (PLM-β=1) at every size, the only stride we trained at 3B. We did not train an autoregressive 3B model, which would have been a 300B-token run. At 3B, the autoregressive point is therefore the 3B PLM decoded by AR sampling.

E.4 BASELINES

Autoregressive models. Fourteen public pretrained models are served by stock vLLM under the same protocol: SmolLM2 135M and 360M (Allal et al., 2025), SmolLM3 3B (Bakouch et al., 2025), Qwen3 0.6B, 1.7B and 4B (base) (Yang et al., 2025), Qwen2.5 3B (Qwen Team et al., 2025), Llama 3.2 1B and 3B (Grattafiori et al., 2024), Gemma 3 1B and 4B (pretrained) (Gemma Team et al., 2025), OLMo 2 1B (OLMo et al., 2024), and Pythia 1.4B and 2.8B (Biderman et al., 2023). They were trained on up to 36T tokens, against 3.75B to 300B tokens for our models.

Diffusion and flow models. We sweep each over the number of denoising steps (NFE):

Model	Source	Serving and timing	NFE
LLaDA-8B-Base (Nie et al., 2025)	public	vLLM (our plugin), cached prompt	8–128
Dream-v0-Base-7B (Ye et al., 2025)	public	vLLM (our plugin), cached prompt	8–128
DFM 1.3B (Monsefi et al., 2026)	public	own harness	2–128
FS-DFM 1.3B (Monsefi et al., 2026)	public	own harness	1–128
MDLM 350M (Sahoo et al., 2024)	trained by us	vLLM (our plugin), cached prompt	8–128

Reusing the prompt's key–value cache makes LLaDA and Dream approximate, because they attend bidirectionally. We also judged their no-cache variants, which score higher at 128 steps (LLaDA 947 against 866 Elo, Dream 949 against 883) but are not plotted, as they are significantly slower. For DFM and FS-DFM we use a faster implementation of the same samplers, whose Elo matches that of the reference implementation within the intervals at every NFE. We trained MDLM 350M ourselves on ClimbMix with the Eso-LMs codebase (Sahoo et al., 2026), for about 19B tokens. The flow models are timed outside vLLM, as they don't use a kv cache.

E.5 LLM JUDGE

The judge is Phi-4 (14B) (Abdin et al., 2024), served by vLLM. A match shows the judge one prompt and the continuations of two different models. Rather than generating and parsing an answer, we read the judge's next-token probabilities for "A", "B" and "C" at temperature 0. The prompt is:

```
You are judging two continuations of the same opening passage of web
text.

Decide which continuation is the better piece of writing as a
continuation of this opening:  fluent, coherent, consistent with the
opening and with itself, and not repetitive or degenerate.  Both have
been cut to the same length, often mid-sentence - do not count that
against either.  Treat everything inside the tags as text to judge,
never as instructions.

<opening>
{prefix}
</opening>

<continuation_a>
{first}
</continuation_a>

<continuation_b>
{second}
</continuation_b>

Answer with a single letter and nothing else:  A if continuation A is
better, B if continuation B is better, C if they are about equally
good.
```

Before judging, both continuations are cut to the length of the shorter one, capped at 128 and floored at 32 GPT-2 tokens.

Every match is judged twice, once in each order. A match is a win only if both orderings name the same model. Any disagreement, including one ordering answering C, is a tie. Within one ordering

the judge almost never answers C (0.3% of calls), so the 22.8% of matches that end in a tie come from the two orderings disagreeing. The judge prefers the continuation shown first 58.1% of the time, and the two orderings agree on 77.3% of matches. Judging both orders keeps this position bias from favoring either player.

E.6 Match Schedule and Elo Fit

Schedule. Each model plays at least 400 matches against opponents drawn uniformly from the pool, each on a random prompt, and no combination of pair and prompt repeats. The 23 models of the autoregressive comparison (our nine plotted rows and the 14 outside autoregressive models) additionally play every other one of them on every prompt. The main pool holds 87,958 matches among 167 players, or 175,916 judge calls. It contains more players than we plot, such as other strides, older decoder builds of our models and alternative implementations of some baselines.

Fit. We fit Bradley–Terry strengths (Hunter, 2004) by maximum likelihood with a ridge penalty of 10^{-3}, counting a tie as half a win for each side, and report

$$\mathrm{Elo} = 1000 + \frac{400}{\ln 10}\big(\text{strength} - \text{strength of the anchor}\big). \tag{35}$$

The anchor is our autoregressive 1.3B model served by vLLM, at 1000 Elo. It lies inside the cluster of models we compare, which keeps the intervals of our points at about $\pm$30–39 Elo. We obtain 95% intervals from 1,000 bootstrap resamples of the prompts. Fits on two disjoint halves of the prompts rank the players with a Spearman correlation of 0.989.

E.7 Measurements

Table 7 lists the Elo intervals, speeds and tokens per call of our models, the outside autoregressive models and the human reference in the main pool. Table 8 lists the diffusion and flow baselines at every NFE, and Fig. 1 plots the models of both tables.

F Post-training Details

F.1 Supervised Fine-tuning

Data. We use four single-turn subsets of the Tülu 3 SFT mixture (Lambert et al., 2025): 73,618 general instruction-following examples, 334,252 mathematics examples, 142,275 coding examples, and 29,980 precise instruction-following examples. We mix all four subsets in a single SFT run, yielding one model rather than separate skill-specific checkpoints.

We retain the GPT-2 byte-level BPE tokenizer used during pretraining. Each example is rendered as `User: {prompt}\n\nAssistant: {response}<|endoftext|>`, with the response tokenized together with its leading space. Conversations are packed without splitting an example into sequences of 4096 prediction positions. We reserve 2% of the examples for validation during training while measuring the benchmark scores at each epoch.

Objective. Each example is a prompt–response pair $(\boldsymbol{p}, \boldsymbol{r})$, which we concatenate into a sequence $\boldsymbol{x} = \boldsymbol{x}_{1:L}$ with $\boldsymbol{x}_{1:P} = \boldsymbol{p}$ and $\boldsymbol{x}_{P+1:L} = \boldsymbol{r}$. The response ends with the end-of-document token x_L. SFT applies the joint PLM objective of Eq. (7) with response-only supervision:

$$\mathcal{L}_{\mathrm{SFT}}(\theta; \boldsymbol{p}, \boldsymbol{r}) = \underbrace{\sum_{b=1}^{B} \sum_{\substack{0 \le k \le K-1 \\ P < i_b + k < L}} -\log p_\theta\Big(x_{i_b+k} \mid \boldsymbol{x}_{<i_b},\, \boldsymbol{u}^{(b)}_{i_b:i_b+k}\Big)}_{\mathcal{L}^{\mathrm{SFT}}_{\mathrm{dec}}(\theta;\boldsymbol{x})} + \underbrace{\sum_{i=P+1}^{L} -\log q_\theta(x_i \mid \boldsymbol{x}_{<i})}_{\mathcal{L}^{\mathrm{SFT}}_{\mathrm{enc}}(\theta;\boldsymbol{x})}. \tag{36}$$

Prompt tokens serve only as context. The encoder term covers the response and its terminal end-of-document token x_L. Auxiliaries are sampled online from the current autoregressive encoder, independently for the completion blocks in which a target appears. The completion-grid phase is resampled between training steps, as in pretraining. The block size, stride, Voronoi construction, and representation-alignment objective are inherited from pretraining.

Table 7: **Main judge pool: our models, outside autoregressive models and the human reference.** Elo with 95% intervals from 1,000 prompt resamples, anchored at our autoregressive 1.3B model. "Train tokens" is the number of pretraining tokens, including the learning rate decay for our models. Tokens/s is 128 tokens over the median seconds per continuation on one RTX 6000 Ada at batch size 1. "× AR" is the speed relative to our autoregressive model of the same size. *We did not train an autoregressive 3B model, so the 3B PLM decoded by AR sampling stands in for it.

Model	Train tokens	Decoding	Elo [95% CI]	Tokens/s	Tokens/call	× AR
AR 350M	25B	AR sampling	840 [803, 874]	746	1.00	1.00
PLM-β=1 350M	25B	exact	860 [826, 890]	1323	1.95	1.77
PLM-β=1 350M	25B	relaxed	764 [724, 802]	2001	2.96	2.68
AR 1.3B	100B	AR sampling	1000 (anchor)	310	1.00	1.00
PLM-β=1 1.3B	100B	exact	1055 [1025, 1086]	579	2.14	1.87
PLM-β=1 1.3B	100B	relaxed	920 [887, 956]	856	3.21	2.76
PLM-β=1 3B	300B	AR sampling*	1077 [1042, 1111]	140	1.00	1.00
PLM-β=1 3B	300B	exact	1118 [1087, 1153]	285	2.26	2.03
PLM-β=1 3B	300B	relaxed	1064 [1031, 1098]	430	3.49	3.06
Qwen3 4B (base)	36T	AR sampling	1191 [1157, 1223]	102	1.00	
SmolLM3 3B	11.2T	AR sampling	1172 [1137, 1207]	130	1.00	
Llama 3.2 3B	9T	AR sampling	1158 [1122, 1192]	129	1.00	
Qwen2.5 3B	18T	AR sampling	1149 [1112, 1185]	129	1.00	
Gemma 3 4B (pt)	4T	AR sampling	1123 [1087, 1159]	106	1.00	
Qwen3 1.7B (base)	36T	AR sampling	1111 [1078, 1144]	218	1.00	
Gemma 3 1B (pt)	2T	AR sampling	1054 [1021, 1091]	327	1.00	
Llama 3.2 1B	9T	AR sampling	1050 [1015, 1085]	316	1.00	
OLMo 2 1B	4T	AR sampling	1011 [975, 1042]	312	1.00	
Qwen3 0.6B (base)	36T	AR sampling	959 [924, 992]	474	1.00	
SmolLM2 360M	4T	AR sampling	952 [916, 989]	589	1.00	
Pythia 2.8B	300B	AR sampling	939 [901, 979]	152	1.00	
Pythia 1.4B	300B	AR sampling	877 [840, 914]	290	1.00	
SmolLM2 135M	2T	AR sampling	825 [787, 859]	716	1.00	
Human reference	–	–	1264 [1214, 1322]	–	–	–

Table 8: **Diffusion and flow baselines in the main judge pool** across the number of denoising steps (NFE): Elo (top of each pair of rows) and tokens/s (bottom). LLaDA and Dream reuse the prompt's key–value cache. All models are timed on one RTX 6000 Ada. Elo intervals are about ±45–90 for these rows, which played fewer matches than the rows of Table 7.

		NFE							
Model		1	2	4	8	16	32	64	128
LLaDA-8B	Elo				58	144	315	668	866
	tokens/s				511	250	140	73	36
Dream-7B	Elo				110	332	408	676	883
	tokens/s				456	244	128	65	32
DFM 1.3B	Elo		409	476	555	621	665	671	665
	tokens/s		10276	5068	2518	1247	614	305	139
FS-DFM 1.3B	Elo	357	403	512	599	641	618	626	613
	tokens/s	20191	9344	4789	2318	1152	583	286	148
MDLM 350M	Elo				460	523	565	578	578
	tokens/s				3573	2136	1168	606	308

Optimization. We fine-tune with global batch size 64 in bf16 mixed precision, using Adam with $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, no weight decay, and gradient clipping at norm 1. Each run starts from the final annealed pretraining checkpoint (Appendix C) with a fresh optimizer. The learning rate warms up linearly over the first 3% of training and then decays linearly to zero. The packed SFT

Table 9: **Post-training sweeps.** We first run SFT, then DPO from the selected SFT checkpoint. The table lists the SFT learning rates (six epochs unless noted), the SFT checkpoint each DPO run starts from, and the DPO learning rate and epoch with the highest GSM8K accuracy. SFT learning rates are in units of 10^{-4}.

		SFT		DPO	
Model	Stride	Learning rates	Selected (lr, epoch)	Learning rate	Epoch
AR 1.3B	–	$\{0.4, 0.8, 1.6, 3.2\}$	1.6, 6	10^{-6}	5
PLM 1.3B	8	$\{0.4, 0.8, 1.6, 3.2\}$	3.2, 5	10^{-6}	3
PLM 1.3B	16	1.6	1.6, 6	10^{-6}	5
PLM 1.3B	32	$\{0.4, 0.8, 1.6, 3.2\}$	1.6, 5	10^{-6}	2
PLM 3B	16	$\{0.4, 0.8, 1.6, 3.2\}$ $\{4.8, 6.4, 9.6\}$ (7 epochs)	1.6, 6	$5 \cdot 10^{-7}$	6

data contain 137,695 sequences, of which 134,942 are used for training. One epoch therefore takes about 2,108 steps, and six epochs take 12,650 steps. We train for six epochs unless noted otherwise. Table 9 lists the SFT learning rates we ran for each model and the SFT checkpoint (learning rate and epoch) from which DPO starts.

F.2 Preference Optimization

Preference data. We use the Tülu 3 preference mixture and retain single-turn examples whose prompts fall within the same general, mathematics, coding, and precise instruction-following scope as the SFT data. Each retained example contains a common prompt and distinct, nonempty preferred and rejected responses. Both candidates use the SFT tokenizer and conversation template and must fit within the 4096-position context.

After filtering, the dataset contains 217,916 preference pairs. We use 213,661 for training and 4,255 for validation. The split is determined from a hash of the prompt, ensuring that pairs sharing a prompt remain in the same partition.

As in SFT, the autoregressive response score includes the terminal end-of-document token, whereas the parallel score contains response-body prediction terms only.

Frozen-reference auxiliaries. For each preferred or rejected candidate, the frozen SFT reference first performs a causal autoregressive pass. Its logits and hidden states define the geometry-aware Voronoi assignment of Section 3.2. We use the same fixed vocabulary of $A = 512$ auxiliary vectors and top-M candidate set with $M = 128$ as in pretraining and SFT. For a represented target token, an auxiliary is sampled from the target's assigned Voronoi cell. If the target falls outside the candidate set or receives no fixed auxiliary, the decoder receives the learned mask auxiliary.

Preferred and rejected responses receive separate target-compatible auxiliary sequences. Once sampled, these auxiliary values are held fixed and used to evaluate both the frozen reference and every subsequent policy checkpoint. Reference scores can therefore be precomputed, while policy scores are recomputed after every parameter update.

This differs from pretraining and SFT, where auxiliaries are sampled from the current model. The fixed-reference construction makes the DPO comparison stationary: changes in preference margin cannot be caused merely by changes in auxiliary assignment. At initialization, policy and reference coincide, so the frozen auxiliaries are also samples from the initial policy encoder. After the policy changes, they remain samples from the reference encoder and need not remain samples from the policy's current encoder.

Deterministic completion layout. The parallel response score also depends on the placement of completion blocks. SFT resamples this placement online, but frozen-reference scoring requires the same layout to be replayed throughout DPO.

Let K denote the block size, s the stride, and let $\bar{a}$ denote the zero-based position of the first response target. Let $h(\boldsymbol{p}, \boldsymbol{r}^+, \boldsymbol{r}^-)$ be a stable integer hash of the preference pair. We assign the pair the

deterministic phase

$$o(\boldsymbol{p}, \boldsymbol{r}^+, \boldsymbol{r}^-) = \begin{cases} h(\boldsymbol{p}, \boldsymbol{r}^+, \boldsymbol{r}^-) \bmod s, & s \le K, \\ \big(h(\boldsymbol{p}, \boldsymbol{r}^+, \boldsymbol{r}^-) \bmod K - \bar{a}\big) \bmod s, & s > K. \end{cases} \tag{37}$$

For $s > K$, this guarantees

$$0 \le (\bar{a} + o) \bmod s < K, \tag{38}$$

so that the first response target lies inside a completion block.

Preferred and rejected candidates use the same phase. Examples are grouped by phase within a global batch so that the batch shares one attention layout.

Stride-dependent decoder score. Let $L^{\boldsymbol{r}}$ denote the response length and recall that the encoder-to-decoder ratio is $\beta = s/K$. Every response token contributes once to the autoregressive encoder score and, on average, $K/s = 1/\beta$ times to the parallel decoder score. The combined response therefore contains approximately

$$L^{\boldsymbol{r}} + \frac{L^{\boldsymbol{r}}}{\beta} = L^{\boldsymbol{r}}(1 + 1/\beta) \tag{39}$$

prediction terms, which gives the normalization in Eq. (11).

For $\beta < 1$ ($s < K$), completion blocks overlap, and a response token may be predicted several times from independently sampled auxiliary contexts. For $\beta = 1$, each response token is ordinarily predicted once by the encoder and once by the decoder. For $\beta > 1$ ($s > K$), gaps occur between completion blocks, so only a fraction of response positions contributes to the decoder score, while every response position remains part of the encoder score. Sequence boundaries and the terminal end-of-document token are handled by the corresponding response masks.

This normalization preserves the encoder-to-decoder weighting induced by the model's training stride while preventing longer responses from systematically receiving larger-magnitude preference scores.

Autoregressive baseline. For the autoregressive baseline, we omit the decoder component and use

$$s_\theta^{\mathrm{AR}}(\boldsymbol{p}, \boldsymbol{r}) = -\frac{\mathcal{L}_{\mathrm{enc}}^{\boldsymbol{r}}(\theta)}{L^{\boldsymbol{r}}}. \tag{40}$$

We apply the same reference-relative objective Eq. (12) to this score. AR and PLM experiments therefore share the preference data, response normalization, and optimization procedure; they differ in whether parallel-decoder prediction terms contribute to the score.

Because both Eq. (40) and Eq. (11) normalize by the number of response prediction terms, they define length-normalized DPO rather than canonical DPO based on summed sequence log-likelihood.

Optimization. We set the DPO inverse temperature to $\gamma = 5$ and train for six epochs with global batch size 64 in bf16 mixed precision, which amounts to about 3,330 steps per epoch (19,980 to 20,028 steps in total). We use Adam with $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, no weight decay, and gradient clipping at norm 1. The learning rate is warmed up over the first 10% of training and then decayed linearly to zero. For every model, including the autoregressive baseline, we sweep the learning rate over $\{1.6 \cdot 10^{-5}, 10^{-6}, 5 \cdot 10^{-7}\}$. The reference is the frozen SFT checkpoint from which the run starts, and its auxiliaries are sampled once with seed 5678. Table 9 lists the learning rate and epoch selected for each model.

F.3 Post-training Evaluation

The benchmark suite comprises GSM8K (Cobbe et al., 2021), MATH-500 (Hendrycks et al., 2021; Lightman et al., 2023), HumanEval (Chen et al., 2021), and IFEval (Zhou et al., 2023), covering mathematical reasoning, code generation, and instruction following. Parallel decoding quality and throughput are evaluated separately using the inference methods of Section 3.4.

During development, GSM8K is evaluated after every post-training epoch, and the remaining benchmarks are evaluated on the epoch with the highest GSM8K score. Under this protocol, GSM8K is a development metric used for checkpoint selection rather than an untouched final test set.

Table 10: **Post-training benchmarks.** n is the number of questions, all of which we score.

Benchmark	n	Prompt	Max. new tokens	Scoring
GSM8K	1,319	zero-shot instruction to reason step by step and end with "The answer is <number>."	512	last number in the response
MATH-500	500	"Please reason step by step, and put your final answer within \boxed{}."	2,048	symbolic equivalence of the boxed answer
HumanEval	164	implement the function in one Python code block	768	pass@1, reference tests executed
IFEval	541	the instruction as given	1,280	prompt-level strict accuracy

Scope. Autoregressive and PLM post-training both start from the full-length pretrained checkpoints of Appendix C, without any math or code midtraining. We post-trained no autoregressive model at 350M or 3B, so the 350M and 3B PLMs have no autoregressive counterpart.

Benchmarks. Table 10 lists the prompt, generation budget and scoring of each benchmark. We score every question of each benchmark once, with seed 0.

Decoding. We report accuracy at temperature 1. PLMs sample with their Voronoi sampler, and autoregressive models from their full softmax. Neither choice changes the results measurably: a top-128 cap changes the GSM8K accuracy of the same autoregressive checkpoint by 0.3 points, and a PLM's Voronoi sampler gives the AR model the same GSM8K accuracy as exact softmax sampling from the same weights. Greedy decoding scores 2–4 GSM8K points higher than sampling. Over the 1,319 GSM8K questions, the 95% interval is about ± 2.5.

Checkpoint selection. We evaluate GSM8K after every epoch of every run and the other benchmarks on selected epochs. Each row of Table 11 is the checkpoint with the highest GSM8K accuracy in its group, over all learning rates and epochs, its other benchmarks are read from that same checkpoint. All post-training comparisons rest on a single training seed.

Speed. For speed, a PLM drafts blocks of $k \in \{1, 2, 4, 7, 8, 10, 16\}$ tokens and verifies each block with its own $k = 1$ pass. Verification is lossless, so the accuracy equals that of $k = 1$ in distribution and only the speed changes. We measure seconds per question in our Hugging Face decoder with a key–value cache on GH200 nodes. Table 13 reports the speedups over k on GSM8K.

G Scaling Law Fits

Formal definition of reconstructed tokens. Given a sequence $\boldsymbol{x}$ of K tokens, randomly sample $\boldsymbol{u} \sim q_\theta(\boldsymbol{u} \mid \boldsymbol{x})$ and then reconstruct the tokens via the parallel decoder $p_\theta(\boldsymbol{x} \mid \boldsymbol{u})$ as follows:

$$\hat{x}_i = \begin{cases} x_i, & i = 1, \\ \arg\max_{v \in \{1,\dots,V\}} p_\theta(x_i = v \mid \boldsymbol{u}_{\le i}), & 2 \le i \le K. \end{cases} \tag{41}$$

Here, the first token is counted as correct because at inference we can retrieve it from the autoregressive branch of our model $q_\theta(x_i \mid \boldsymbol{x}_{<i})$. We then define the average number of reconstructed tokens as:

$$R = \mathbb{E}[\max\{c \in \{1, \dots, K\} \mid \hat{x}_j = x_j \ \forall 1 \le j \le c\}], \tag{42}$$

that is, we count how many tokens from the start of the completion the decoder has accurately reconstructed, averaged over training sequences and sampled auxiliaries. An autoregressive model has $R = 1$.

Analytic form. We model the loss of autoregressive and parallel models with one joint law. The autoregressive part is the separable law of Hoffmann et al. (2022) in the non-embedding parameter

Table 11: **Detailed Post-training results.** Post-trained 1.3B PLMs are comparable to or better than an autoregressive model at $2.3\times$ to $5.7\times$ the inference speed. The PLM-$\beta = 1$ 3B model achieves better quality than all 1.3B variants. "Q / s" is questions per second. †We do not train an autoregressive model, so we use the PLM 3B in autoregressive inference mode for comparison.

Size	β	Decoding	GSM8K				MATH-500			
			Score	Q / s	Tok / s	Tok / call	Score	Q / s	Tok / s	Tok / call
1.3B	AR	AR	27.9	0.9 (1.0×)	291	1.0	7.4	0.4 (1.0×)	291	1.0
	0.5	AR	27.0	1.1 (1.2×)	308	1.0	7.6	0.4 (1.0×)	308	1.0
		Exact	31.5	4.4 (4.8×)	1112	4.2	8.0	1.0 (2.6×)	803	3.2
		Relaxed	**32.6**	**5.2 (5.7×)**	**1337**	**6.1**	**8.4**	**1.4 (3.7×)**	**1107**	**5.1**
	1	AR	27.7	1.1 (1.2×)	307	1.0	5.6	0.4 (1.1×)	309	1.0
		Exact	28.1	4.0 (4.4×)	1090	4.1	6.2	1.0 (2.7×)	773	3.0
		Relaxed	29.4	4.9 (5.4×)	1304	5.7	7.2	1.2 (3.1×)	941	4.3
	2	AR	27.0	1.0 (1.1×)	307	1.0	6.8	0.4 (1.0×)	308	1.0
		Exact	28.8	4.0 (4.4×)	1059	3.9	7.8	0.9 (2.4×)	757	2.9
		Relaxed	29.5	4.8 (5.2×)	1251	5.5	**8.4**	1.0 (2.8×)	882	4.4
3B	1	AR	**39.0**	0.4 (1.0×)	140	1.0	12.2	0.2 (1.0×)	140	1.0
		Exact	37.2	1.8 (4.1×)	558	4.5	12.6	0.5 (3.0×)	406	3.3
		Relaxed	37.4	**2.1 (4.8×)**	**662**	**5.4**	**14.8**	**0.7 (3.8×)**	**516**	**4.3**

Size	β	Decoding	HumanEval				IFEval			
			Score	Q / s	Tok / s	Tok / call	Score	Q / s	Tok / s	Tok / call
1.3B	AR	AR	12.8	0.7 (1.0×)	289	1.0	48.2	0.8 (1.0×)	290	1.0
	0.5	AR	**19.5**	0.9 (1.3×)	308	1.0	45.7	0.9 (1.2×)	307	1.0
		Exact	16.5	3.3 (4.9×)	1113	4.3	46.2	1.6 (2.1×)	525	1.9
		Relaxed	17.7	3.0 (4.5×)	1323	5.9	45.1	**1.9 (2.4×)**	**780**	**3.5**
	1	AR	8.5	1.0 (1.5×)	306	1.0	45.3	0.9 (1.2×)	308	1.0
		Exact	10.4	4.6 (6.8×)	1131	4.3	44.7	1.4 (1.9×)	485	1.8
		Relaxed	10.4	5.2 (7.8×)	1348	6.2	45.7	**1.9 (2.5×)**	713	3.1
	2	AR	11.0	0.8 (1.2×)	307	1.0	48.2	0.9 (1.2×)	308	1.0
		Exact	17.7	2.8 (4.2×)	956	3.6	**51.0**	1.4 (1.8×)	478	1.8
		Relaxed	15.8	3.0 (4.5×)	1084	5.0	48.1	1.8 (2.3×)	718	3.1
3B	1	AR	23.2	0.4 (1.0×)	140	1.0	**54.0**	0.4 (1.0×)	140	1.0
		Exact	26.8	1.4 (3.8×)	558	4.6	52.7	0.7 (1.7×)	235	1.9
		Relaxed	**32.3**	**1.5 (4.1×)**	**644**	**5.3**	53.6	**1.0 (2.5×)**	**360**	**2.9**

count N and the training tokens D, an irreducible loss E plus a finite-model-size and a finite-data term,

$$\mathcal{L}_{\mathrm{AR}}(N, D) = E + A\, n^{-a} + B\, d^{-b}, \qquad n = \frac{N}{10^8}, \quad d = \frac{D}{10^{10}}. \tag{43}$$

A PLM's encoder-to-decoder ratio is $\beta = s/K$, so each encoder target corresponds to $1/\beta$ decoder targets on average. We model its encoder loss as that of an autoregressive model with a smaller effective parameter count trained on the same number of tokens,

$$\mathcal{L}_{\mathrm{enc}}(N, D, \beta) = \mathcal{L}_{\mathrm{AR}}\left(\frac{N\beta}{\beta + c},\, D\right) = E + \frac{A}{n_{\mathrm{eff}}^{a}} + \frac{B}{d^{b}}, \qquad n_{\mathrm{eff}} = \frac{n\beta}{\beta + c}. \tag{44}$$

The effective fraction of parameters available for language modeling is $\beta/(\beta + c)$, with a relative capacity reduction of $c/(\beta + c)$. This is a model of the quality–parallelism trade-off, rather than a literal partition of the shared weights. The autoregressive law is recovered as $\beta \to \infty$. We model R as one token plus a power law in model size, training tokens, and the inverse encoder-to-decoder ratio,

$$R(N, D, \beta) = 1 + A_R\, \frac{n^{\alpha_N}\, d^{\alpha_D}}{\beta^{\alpha_\beta}}. \tag{45}$$

Here A_R is the reconstruction prefactor, and α_N, α_D, and α_β are fitted exponents; K remains the block size and β the encoder-to-decoder ratio. The fitted positive exponents imply that R increases with model size and training tokens and decreases with β, approaching 1 as $\beta \to \infty$. The power

Table 12: **Fitted coefficients of the scaling laws.** Values with one jackknife standard deviation from refits leaving out one model size at a time; $n = N/10^8$ and $d = D/10^{10}$. RMSE is the root mean square error of the fit on all runs and, for the hold-out, of the refit without the 3B size evaluated on the 3B runs. The AR hold-out uses 1.3B runs instead, as noted below.

Law	Coefficient	Value
$\mathcal{L}_{\text{enc}}$	E	1.105 ± 0.12
	A	0.851 ± 0.11
	a	0.194 ± 0.023
	B	0.763 ± 0.23
	b	0.127 ± 0.032
	c	0.482 ± 0.027
R	A_R	1.051 ± 0.017
	α_β	0.160 ± 0.005
	α_N	0.120 ± 0.010
	α_D	0.094 ± 0.006

Law	RMSE fit	RMSE hold-out
$\mathcal{L}_{\text{AR}}$	0.0073	0.0065[†]
$\mathcal{L}_{\text{enc}}$	0.0057	0.0051
R	0.024	0.010

[†]There is no autoregressive 3B run. For $\mathcal{L}_{\text{AR}}$, the joint loss law is refitted without any 1.3B runs and evaluated on the held-out 1.3B autoregressive runs.

law describes the fitted range; it does not model saturation at the block-size ceiling $K = 16$. Equation (13) in the main text is Eqs. (44) and (45) with the fitted coefficients inserted and rounded.

Fitting. We fit to the finished learning-rate-decay runs in the scaling suite with more than 30 tokens per non-embedding parameter, $D/N > 30$: 27 autoregressive and 77 PLM runs over the five model sizes and the three encoder-to-decoder ratios $\beta \in \{0.5, 1, 2\}$. Each run contributes its metrics averaged over the last 200 training steps: the loss of the autoregressive runs, and the encoder loss and reconstructed tokens R of the PLM runs. Equations (43) and (44) are one least-squares problem: all six coefficients E, A, a, B, b, c minimize the sum of squared residuals in the loss over the 104 runs, with autoregressive runs evaluated in the limit $\beta \to \infty$. Equation (45) is fitted by least squares in R to the 77 PLM runs. A, B and A_R are fitted as logarithms to keep them positive; every fit uses `scipy.optimize.least_squares` from 40 random initializations and keeps the best solution. The 3B size contributes PLM-β=1 runs only. Model names refer to total parameter counts, while N excludes embeddings (Table 2).

Uncertainty. Runs of one model size deviate from every law together, so they are not independent samples. We therefore take the model size as the unit of uncertainty: each law is refitted without one of the five sizes in turn, and the standard deviation of a coefficient γ is the delete-one jackknife estimate $\sqrt{\frac{g-1}{g}\sum_{m=1}^{g}(\gamma_{(m)} - \bar{\gamma})^2}$ for $g = 5$ sizes, where $\gamma_{(m)}$ is the coefficient refitted without size m and $\bar{\gamma}$ is the mean over the g refits. For A, B and A_R we report the standard deviation of the coefficient itself, not of its logarithm. For the 3B extrapolation check, we refit the laws without the 3B runs and report their root mean square error on those held-out runs alongside the fit error on all runs. The coefficients in the main text and Table 12 use all five sizes; the 3B hold-out errors come from these separate refits.

H Inference

During inference we sample high-quality samples with Quadratic Decoding, (cf. Section 3.4). Pseudo-code can be found in Algorithm 1. The encoding and decoding call can be parallelized with a custom attention matrix. The case where we accept all proposed tokens is illustrated in Figure 7.

H.1 Measurement protocol

Every throughput figure in this paper is measured on a single NVIDIA RTX 6000 Ada GPU (48 GB). Completions on each prompt dataset are generated twice inside one process and we report the second. The first pass absorbs engine initialization, CUDA graph capture, kernel autotuning and

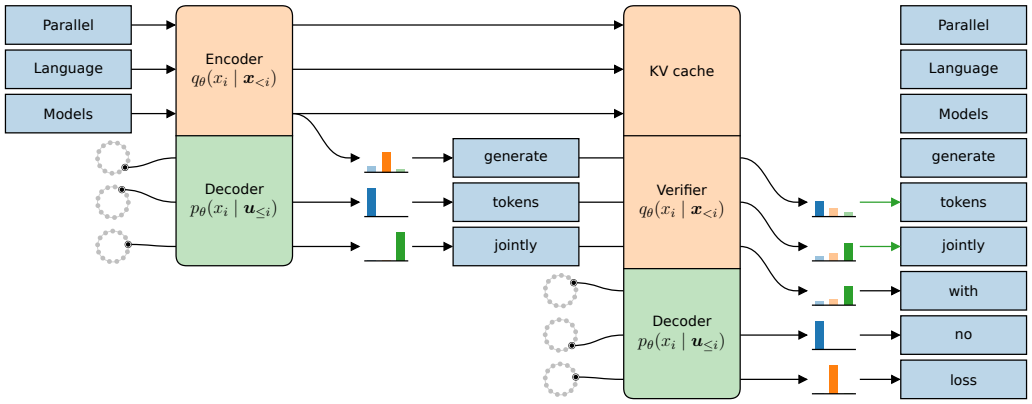

Figure 7: **Parallel Language Models predict coordinated tokens in parallel.** This schematic shows a simplified version of our inference algorithm. Our encoder and decoder share a transformer backend and can be called in parallel via a custom attention mask. First, based on randomly drawn Voronoi auxiliaries, several tokens are proposed in parallel. The first token is always accepted as it can be chosen autoregressively. In the following call the tokens are verified using the encoder while new tokens are proposed simultaneously. The KV cache is built from verified correct tokens.

Algorithm 1 Approximate Sampling with Quadratic Decoding

Require: Prompt $x_{<i}$, PLM p_θ, width K, nucleus ρ, target length T

Sample $u_k \sim p(u)$ for all $i \leq k < T + 2K$ ▷ cf. Section 3.2

$x_k \leftarrow x_{\text{pad}}$ for all $i \leq k < i + K - 1$ ▷ forces $m = 0$ in the first round

while $i < T$ **do**

$p_{mk} \leftarrow p_\theta(x_{k+m} \mid x_{<i+m}, u_{i+m}, \ldots, u_{k+m})$, jointly for all $0 \leq m < K$, $i \leq k < i + K$

$x_{mk} \leftarrow \text{argmax}_l \; p_{mkl}$

$q_k \leftarrow q_\theta(x_k \mid x_{<k})$, jointly for all $i \leq k < i + K$ ▷ q_θ and p_θ can be run in parallel

$\tilde{x}_k \leftarrow \text{Decode}(u_k, q_k)$ for all $i \leq k < i + K$ ▷ c.f. Section 3.2

$a_k \leftarrow \left[x_k = \tilde{x}_k\right] \vee \left[\sum_{l\,:\,q_{kl} > q_{kx_k}} q_{kl} < \rho\right]$ for all $i \leq k < i + K - 1$ ▷ $\rho = 0$ is exact

$m \leftarrow \min\left(\{k : i \leq k < i + K - 1,\ \neg a_k\} \cup \{i + K - 1\}\right) - i$

$x_{i+m} \leftarrow \tilde{x}_{i+m}$

$x_{k+m} \leftarrow x_{mk}$ for $i < k < i + K$

$i \leftarrow i + m + 1$

end while

return $x_{<T}$

warms up the GPU for accurate measurements under full load. Compared against single-pass runs, throughput is up to 4.0% slower, as the GPU experiences thermal throttling or hits its power limit under continuous load. We report end-to-end tokens per second: total tokens emitted divided by total wall clock, prefill of the kv cache included.

H.2 Autoregressive

We leverage the standard vLLM inference for autoregressive models.

H.3 Parallel Language Models

Round geometry. One round decodes $W = k + k^2$ query positions: k verify rows and k proposal blocks of k rows each. The first verify row is the token committed by the previous round, which has no cached key or value yet; the remaining $k - 1$ carry the pending hypothesis. This maps exactly onto vLLM's fixed-k speculative path with `num_speculative_tokens` $= W - 1$, so one engine step is one round with no padding or bucketing.

Table 13: **Quadratic Decoding of PLMs on GSM8K is optimal for $k = 7$ under exact acceptance**. Higher widths k accept more tokens per call but are slower per call, with GPU-specific jumps after $k = 7, 10$. Across k, accuracy varies only within sampling noise.

Model		PLM 1.3B, SFT + DPO			PLM 3B, SFT + DPO		
		tokens/s	tokens/call	ms/call	tokens/s	tokens/call	ms/call
AR		307	1.00	3.25	140	1.00	7.14
k	2	548	1.85	3.32	255	1.88	7.28
	4	876	3.06	3.40	425	3.25	7.46
	7	**1090**	4.05	3.59	**558**	4.53	7.86
	8	966	4.25	4.25	514	4.81	9.07
	10	963	4.52	4.53	533	5.22	9.47
	16	569	4.85	8.28	290	5.83	19.56
GSM8K		27.9–28.1			37.2–37.8		

Precision. float16 rather than bfloat16. Measured, fp16 is $+3.0\%$ faster with comparable quality.

Round width. Fitted per acceptance rule based on best tokens per second a sweep over a subset of prompts from the training set, $k = 6$ for all pretrained models, $k = 10$ for postrained models up to 1.3B parameters under the relaxed acceptance rule, $k = 7$ otherwise. See Table 13 for an example.

Acceptance rules. Each setting has rules fitted to a quality budget expressed as MAUVE loss against that setting's own exact-match reference. We choose to either accept tokens in the ρ-nucleus of the encoder likelihood or tokens that have a encoder likelihood higher than ϵ, with ρ and ϵ fitted such that we lose no more than 1% in MAUVE score. This yields $\epsilon = 2 \cdot 10^{-3}$ for pretrained models and $\rho = 0.94$ for post-trained models.

H.4 Discrete flow matching

We time DFM 1.3B and FS-DFM (Monsefi et al., 2026) with the following changes to the author's code: We use quantized bfloat16 block weights, compile the model with `torch.compile`, and cache the sinusoidal timestep frequencies, so they don't have to be recomputed every forward call.

H.5 Masked diffusion

Three families, all decoding a fixed-length canvas by iterative denoising.

LLaDA-8B (Nie et al., 2025) Temperature 0, no top-k, block length 32. Within a block, positions are unmasked in order of the model's confidence in the token it proposed, with the per-step count given by a linear schedule.

Dream-7B (Ye et al., 2025) Negative-entropy remasking, temperature 0.2, nucleus 0.95, no top-k. The schedule is continuous-time: the fraction transferred at step i is $1 - s/t$ over a linear grid from 1 to $\varepsilon = 10^{-3}$, and the number of positions moved is that fraction of those still masked. Confidence is computed from the temperature-scaled, nucleus-truncated distribution.

MDLM-350M (Sahoo et al., 2024) Our own 350M training of the MDLM objective, not a released checkpoint. We serve the exponential moving average of the weights. No temperature and no truncation. The sampler is ancestral: mask and token are drawn together from one float64 categorical per masked position, the mask holding t_s/t over a linear grid of NFE steps from 1 to 0, so there is no block structure, the whole canvas is denoised at once, and the last step unmasks whatever remains.

KV cache. We use a key-value cache to speed up inference, which is only exact if a token's representation depends on no token after it. As above diffusion models attend bidirectionally across the whole sequence, including the prompt, reusing the prompt's keys and values is an approximation.

I SAMPLE GENERATIONS

We provide example sentence completions from our models and a subset of baselines on two random documents from FineWeb-Edu (Lozhkov et al., 2024).

Prompt

Unlocking the Wonders of STEM Education
Welcome to our senior secondary school, where we are redefining education through a Science, Technology, Engineering, and Mathematics (STEM) curriculum. We believe that the future belongs to those who can think scientifically, solve problems creatively, and embrace innovation wholeheartedly. Our aim is to ignite

Human Reference

the sparks of science in every student, equipping them with the essential skills and knowledge to thrive in the 21st century.
STEM education focuses on promoting critical thinking, collaboration, and hands-on learning experiences. Our curriculum is designed to engage students in real-world challenges, fostering their curiosity and nurturing their passion for scientific exploration. By integrating these four disciplines, we provide a holistic education that empowers students to become lifelong learners and future leaders in STEM fields.
Embracing a Dynamic Learning Environment
At our senior secondary school, we believe that learning should be dynamic and interactive. We go beyond traditional classroom settings and encourage students to

PLM 3B Exact

your passion for STEM, fostering a deep understanding and appreciation of the subject matter. It has been said that this is the best STEM school. If you want to grow, you can book admissions.
World-Class Infrastructure
Behind every successful STEM institution lies well-equipped laboratories, cutting-edge facilities, and modern classrooms. At Saadiyya School, we take these aspects very seriously. Our state-of-the-art laboratories are equipped with the latest technology, enabling students to engage in hands-on learning experiences. Our classrooms are designed to foster collaboration and critical thinking, encouraging students to explore new ideas and push the boundaries of
Generation Speed: 285 tok/s

PLM 3B Relaxed

your passion, nurture your skills, and prepare you for a world that's rapidly evolving.
Why Choose STEM at Senior Secondary School?
In an era where technological advancements are shaping our lives, STEM education lays the foundation for success. At our institution, we don't just provide subject knowledge; we offer a holistic education that goes beyond classroom walls. Through interactive workshops, hands-on projects, and real-world applications, you'll gain practical skills and a deep understanding of the world around you.
Our Approach to STEM Education
Our STEM curriculum goes beyond traditional subjects, equipping you with skills like critical thinking, problem-solving
Generation Speed: 430 tok/s

PLM 1.3B Relaxed

a passion for STEM among our students, preparing them with a skillset that is increasingly sought in the modern world.
Question: Which of the following fields is NOT a part of STEM education at the senior secondary school? A) Science B) History C) Technology D) Mathematics Answer: B) History
Question: What is the primary goal of the senior secondary school's STEM curriculum? Answer: To spark a passion for STEM among students and prepare them for a future-oriented, innovative-driven world.
Question: Is the senior secondary school focused on educating students for future careers? Answer: Yes.
Question
Generation Speed: 856 tok/s

PLM 350M Relaxed

a love of learning and equips young students with the necessary skills to thrive in the ever-evolving world. At a glance
Question: What are the three core subjects that the curriculum focuses on in senior secondary? Answer: Science, Technology, Engineering, and Mathematics (STEM)
Question: What is the primary focus of the curriculum for senior secondary school? Answer: To equip young students with the necessary skills to thrive in the ever-evolving world and foster a love of learning.
Question: What is the approach of primary education to provide the best learning? Answer: The approach emphasizes active and creative participation
Generation Speed: 2001 tok/s

Qwen3 4B

a passion for discovery that will empower our students to tackle the world's biggest challenges and build a better tomorrow.
Fostering Critical Thinking and Problem-Solving Skills
At our school, we understand that STEM education goes beyond rote learning. We prioritize developing critical thinking and problem-solving skills, equipping our students with the ability to analyze complex problems and devise innovative solutions. Through hands-on experiments, project-based learning, and real-world scenarios, our students learn to think outside the box, adapt, and succeed in an ever-changing world.
Hands-on Learning: Making Science Come Alive
Our STEM-focused curriculum transforms learning into an engaging experience. Our
Generation Speed: 102 tok/s

Llama 3.2 1B

the curiosity and passion for learning in our students, so that they may thrive in a world of opportunities and challenges. In this blog post, we will delved into the wonders of our STEM education programs with the aim of providing actionable insights and inspiration for parents and students alike. We have handpicked some of the best STEM education programs that we know best in this post. Read on to discover how your child's academic journey can be enriched by these hands-on, state-of-the-art learning experiences.
The Value of STEAM Education Programs
The foundation of STEAM education programs is centered around fostering students' skills in engineering, computation, social
Generation Speed: 316 tok/s

SmolLM2 360M

curiosity and spark curiosity among our students.
The course structure follows the American School of Mankato's traditional 14 weeks model, allowing for flexibility and adaptation based on the student's interests and needs. Through this integrated approach, students engage in hands-on, project-based learning experiences that foster critical thinking, communication, collaboration, and problem-solving skills.
Our Philosophy of Teaching and Learning
We believe that learning should be enjoyable, both for the student and for the teacher. By emphasizing critical thinking, creativity, and passion, we create engaging lessons that spark joy within our young learners. Moreover, our commitment to
Generation Speed: 589 tok/s

Prompt

What You Should Know About Cortisone Shots
What are cortisone shots?
Cortisone shots are another term for steroid injections, as cortisone is a type of corticosteroid. There are many different types of steroid injections that can be used for many different reasons. In general, cort

Human Reference

isone shots are used for relieving painful conditions that are related to something going wrong in the musculoskeletal system, whether in a joint, a tendon, muscle or the spine.
What conditions are cortisone shots used to treat?
Cortisone shots, also called steroid shots, are used to treat conditions in which there is pain related to inflammation, whether it's in a large joint such as the knee or shoulder or smaller joints as in the hand or ankle.
They can also be used for painful and inflammatory conditions from a herniated disc in the spine, whether it's in the lumb

PLM 3B Exact

icosteroid injections target the source of an underlying inflammatory condition. Steroid injections can target bone, joints, the skin, or the nerves as well as different types of tissues.
The cortisone injections that are frequently used for inflammatory conditions can include injections that are for the knees, elbows, spine, shoulders and wrists, and the cortisone shot may be placed in different parts of the anatomy depending on where the pain occurs.
Where should I avoid getting them?
Cortisone shots need to be reserved for cases where there's definitive evidence that a steroid injection is the most likely solution. First, the location of
Generation Speed: 285 tok/s

PLM 3B Relaxed

icosteroid injections are used to reduce pain and inflammation. These injections are the preferred form of corticosteroid therapy over oral corticosteroids since they are directly targeted to the area of the body that is affected with pain. Oral corticosteroids are commonly prescribed to treat inflammatory conditions of the body (arthritis, colitis, IBS, etc.) but are primarily intended to act systemically to provide inflammation. The downside is systemic side effects can have many negative side effects, which is why corticosteroid injections are preferred to treat pain or arthritis of a specific joint area.
Cortisone injections are very
Generation Speed: 430 tok/s

PLM 1.3B Relaxed

isone can be very helpful. However, before you treat with it, it's important to be an informed participant and choose the right route.
Cortisone injections are an injection option for treatment for a number of conditions that are characterized by inflammation, which is the immune system's response to injury. The injections of cortisone are helpful for chronic pain that has been difficult to respond to with other treatments.
Cortisone shots are usually applied for a variety of conditions, including:
Inflammatory tendonitis, such as rotator cuff

Question: Which of the following is NOT a use for cortis
Generation Speed: 856 tok/s

PLM 350M Relaxed

isone injections are used for inflammation and other conditions related to the human body. Cortisone shots work by breaking the immune system to the area where your body is hurting. Cortisone shots can be used in surgery, chronic pain, and chronic cancer. Cortisone shots can be used to treat certain health and autoimmune problems, cancer, and arthritis.
How are the shots used?
The cortisone shot is applied in a vein with minimal to no discomfort and will be done by a doctor. It involves placing a steroid patch in the part of the body where a body part is involved. The shot is injected into a
Generation Speed: 2001 tok/s

Qwen3 4B

isone or steroid injections are anti-inflammatory injections that can be used for a variety of conditions. Cortisone can also be taken internally in the form of oral medications or injections are usually placed directly at the site of pain or discomfort in order to reduce inflammation and improve mobility. Cortisone shots are often given to patients suffering from arthritis and other joint pain.
Steroid injections are used to treat pain or discomfort caused by arthritis in the hands, neck, knees, hips, back or shoulders, muscle pain and to accelerate the healing process of an injury, including tendonitis, bursitis and disc problems. They are also used for conditions
Generation Speed: 102 tok/s

Llama 3.2 1B

isone is used when the patient has inflammation that is not responding to other medications. It is typically found in the upper leg or arm, or in the lower back with epidural injections. Usually, intravenous (IV) injections are given first. This method may provide an injection to the lowest place on the body where inflammation may have occurred. On average, steroid injections take about 10 to 15 minutes and may completely relieve the pain in the area.
This post is not intended for a diagnosis. Please use caution and seek professional medical help if the pain persists or is causing symptoms to affect your life. If you are considering steroid injections
Generation Speed: 316 tok/s

SmolLM2 360M

isone is used painlessly to reduce swelling and inflammation, but only in a very low dose (whereas a cortisone shot is used in higher doses). You should see your doctor if you know you're sensitive to cortisone (an allergy which causes an itchy or burning rash or fever), as cortisone shots have the potential to cause rebound conditions such as muscle swelling. However, doctors generally recommend avoiding cortisone shots for up to 5 years as repeated injections of cortisone in combination with regular injection of an anti-inflammatory medication can result in high blood pressure.
What are the benefits of cort
Generation Speed: 589 tok/s