IMLE-LLM: LATENT-CONDITIONED WINDOWED DECODING FOR FASTER INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Autoregressive language models emit one token per forward pass, so generation latency grows linearly with output length. Writing several tokens per pass would cut this latency, but tokens written in the same pass cannot see one another. Most existing methods therefore recover the missing dependence outside the pass, with a draft model and a verifier or with extra passes. Yet tokens of one pass can all see the same random draw. We propose IMLE-LLM, a fine-tuning recipe that can turn any pretrained decoder-only transformer into a windowed decoder, adding only a small latent projection. All positions of a window share one Gaussian latent, so a single forward pass commits a window of mutually dependent tokens, with no draft model, verifier or extra passes. We fit this implicit model with conditional implicit maximum likelihood estimation (conditional IMLE), which needs no network that infers the latent. Built on Qwen3-8B, IMLE-LLM reaches a mean speed-up of $3.39\times$ over one-token decoding while keeping most of the accuracy of an autoregressive fine-tune.

1 INTRODUCTION

Large language models now write long answers, such as step-by-step solutions to math problems and complete programs, one token at a time. Each token needs its own forward pass through the model, and each pass must wait for the one before it, so an answer of n tokens costs n passes in a row (Leviathan et al., 2023). When a model serves one request at a time, each pass is limited by memory bandwidth rather than by computation, so a pass over a few more positions takes about as long as a pass over one (Chen et al., 2023). Writing several tokens in each pass would therefore cut the time a user waits for an answer.

The difficulty is that tokens written in the same pass cannot see one another. Each position must pick its token without knowing what the other positions pick. Suppose the next two tokens should read "Los Angeles" with probability 0.4, and "San Diego" or "San Jose" with probability 0.3 each. If each position takes its own most likely token, we get "San" (probability 0.6) followed by "Angeles" (probability 0.4). The result, "San Angeles", was never a possible answer. More training does not fix this. A model that predicts the tokens of a pass independently can at best match the distribution

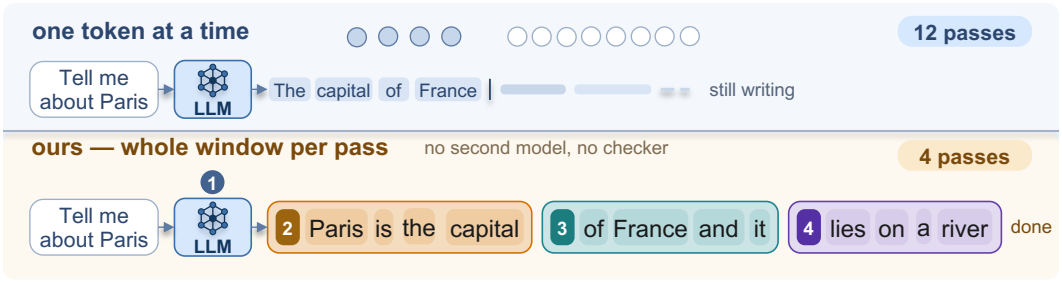


Figure 1: Decoding one token at a time (top) takes twelve passes for twelve tokens. IMLE-LLM (bottom) reads the prompt in one pass and then writes four tokens per pass, four passes in all.

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.

of each token on its own, and it loses how the tokens go together (Gu et al., 2018; Huang et al., 2022).

Most existing methods work around this barrier by taking the dependence from outside the pass. Speculative decoding lets a small drafter propose several tokens, and the large model keeps those it accepts after checking them in one pass (Leviathan et al., 2023; Chen et al., 2023; Li et al., 2025; Chen et al., 2026). This can keep the large model's output unchanged. A second line of work, to which ours belongs, trains the model itself to write several tokens in each pass, with no drafter. It uses multi-token prediction (Kirchenbauer et al., 2026; Gat et al., 2025) or diffusion language models adapted from autoregressive ones (Cheng et al., 2026; Wu et al., 2026; Liu et al., 2026a). These models still predict the tokens of a pass independently (Kirchenbauer et al., 2026; Arriola et al., 2025). To avoid mistakes like the one above, they typically commit only the tokens they are confident about and leave the rest to later passes (Wu et al., 2026; Liu et al., 2026a). Both lines thus pay for the dependence outside the pass, with a drafter and a check or with extra passes.

This raises a simple question. Can a single pass produce tokens that depend on one another, without either cost? Tokens written in the same pass cannot see one another, but they can all see the same random draw. Suppose every position written in a pass receives the same random vector, drawn once before the pass. Given this vector, the positions are still computed independently, so the pass stays parallel. Across draws, the positions change together, so the tokens depend on one another through the vector. In the example, a model of this kind can put almost all its probability on the three real cities. Under each draw, the most likely tokens at the two positions then spell one city. The idea is general. Any joint distribution over the tokens of a pass can be written as an average of distributions that treat the tokens independently (Hayakawa et al., 2025), so one pass with a shared random vector can, in principle, capture any dependence among its tokens.

With the shared draw as an extra input, a pretrained language model becomes an implicit model of its next window. Instead of predicting each position on its own, it turns one random draw into a window. We can draw windows from it, but the probability it gives a window is an average over draws with no closed form. Latent-variable models of parallel text usually train an extra network that infers each example's latent (Kaiser et al., 2018; Ma et al., 2019; Shu et al., 2020; Xie et al., 2026). Conditional implicit maximum likelihood estimation (conditional IMLE) needs no such network, since it fits implicit models from random draws of the latent and the training data alone (Li et al., 2020). The dependence is thus learned inside the model, not paid for outside the pass.

Our method, IMLE-LLM, brings these pieces into a pretrained autoregressive model (Figure 1). We append W input positions for the next window, give each the same projection of one Gaussian latent, and fine-tune the model with conditional IMLE to write all W tokens in one forward pass (Section 3). The only new parameters are the projection matrix ($\sim$0.5M), under 0.01 % of those in Qwen3-8B (Yang et al., 2025). Each window then takes one draw and one forward pass, with no drafter, no check and no extra passes.

We fine-tune Qwen3-8B with windows of $W = 4$ tokens and test it on six benchmarks in math, code and instruction following (Section 4). IMLE-LLM reaches a mean speed-up of 3.39$\times$ over one-token decoding of the original Qwen3-8B while keeping 97.3 % of the math accuracy and 95.2 % of the code accuracy of an autoregressive fine-tune on the same data. Against recent diffusion language models built on the same Qwen3-8B, IMLE-LLM is more accurate and faster on the reported benchmarks (Section 4.2).

Our contributions are as follows.

- We introduce IMLE-LLM, which lets a pretrained autoregressive model write a window of mutually dependent tokens in each forward pass through one shared Gaussian latent, a design that fits any decoder-only transformer (Section 3.1).
- We train this implicit model with conditional IMLE, which needs no network that infers the latent, so the only addition to the pretrained model is one projection matrix (Section 3.2).
- We show on Qwen3-8B that a single pass per window is enough in practice, reaching a speed-up above 3$\times$ while keeping over 95 % of the math and code accuracy of an autoregressive fine-tune (Section 4).

2 PRELIMINARIES

2.1 BLOCK-AUTOREGRESSIVE GENERATION

Our method builds on two objects from earlier work, block-autoregressive generation (Wang et al., 2018; Arriola et al., 2025) and conditional implicit maximum likelihood estimation (Li et al., 2020). Given a prompt $\mathbf{c}$, an autoregressive language model factorises a response $\mathbf{y} = (y_1, \ldots, y_n)$ as $p(\mathbf{y} \mid \mathbf{c}) = \prod_{t=1}^{n} p(y_t \mid \mathbf{c}, y_{<t})$, where $y_{<t}$ is the response before position t. Decoding that product one factor at a time costs one forward pass per generated token.

A block-autoregressive model p_θ, also called semi-autoregressive, preserves the autoregressive order across blocks and relaxes it inside each block (Wang et al., 2018; Arriola et al., 2025). It generates $\mathbf{y}$ as B successive blocks of at most W tokens and factorises $\mathbf{y}$ as

$$p_\theta(\mathbf{y} \mid \mathbf{c}) \;=\; \prod_{b=1}^{B} p_\theta\big(\mathbf{y}^{(b)} \mid \mathbf{c}_b\big), \qquad \mathbf{c}_b = \big(\mathbf{c}, \mathbf{y}^{(<b)}\big), \tag{1}$$

where $\mathbf{y}^{(b)}$ holds the tokens of block b and $\mathbf{c}_b$ is the prompt with the blocks before it (Wang et al., 2018; Arriola et al., 2025).

Each factor of Equation 1 is a joint distribution over the positions of its block. Block models differ in how they build that joint. The semi-autoregressive Transformer of Wang et al. (2018) predicts those positions in parallel and independently of one another given $\mathbf{c}_b$, so its block factor is a product of per-position distributions.

Instead, keeping the dependence inside a block needs a mechanism that couples its positions. The block-diffusion family couples them with a discrete denoising diffusion model run inside the block (Arriola et al., 2025). Each denoising step predicts the positions of the block independently given $\mathbf{c}_b$ and the block's current state, so any dependence among them enters only across steps. A block model therefore defines its own distribution over responses, and its samples can differ from an autoregressive model's.

2.2 CONDITIONAL IMPLICIT MAXIMUM LIKELIHOOD ESTIMATION

Conditional IMLE (Li et al., 2020) extends IMLE, the estimator of Li & Malik (2018), to a sampler T_θ, with parameters θ, that maps an input $\mathbf{x}$ and a latent $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ to a sample. The sampler is an implicit model, so its likelihood has no closed form and maximising it directly is intractable.

For N training pairs $(\mathbf{x}_i, \mathbf{y}_i)$ of input and output, conditional IMLE draws m latents $\mathbf{z}_{i,1}, \ldots, \mathbf{z}_{i,m}$ per pair. The estimate $\hat{\theta}$ minimises the expected sum, over all pairs, of the distance $\mathcal{L}$ from $\mathbf{y}_i$ to the nearest of the m samples indexed by j,

$$\hat{\theta} \;=\; \arg\min_{\theta} \; \mathbb{E}_{\mathbf{z}_{1,1},\ldots,\mathbf{z}_{N,m} \sim \mathcal{N}(\mathbf{0},\mathbf{I})} \Big[\sum_{i=1}^{N} \min_{j \in \{1,\ldots,m\}} \mathcal{L}\big(T_\theta(\mathbf{x}_i, \mathbf{z}_{i,j}), \mathbf{y}_i\big) \Big]. \tag{2}$$

The samples of a pair are matched against that pair alone, because samples for another input come from another distribution (Li et al., 2020). Equation 2 encourages a generated sample near every pair, so no pair is ignored, which is how IMLE avoids mode collapse (Li et al., 2020). Li & Malik (2018) show IMLE to be equivalent to maximising likelihood under conditions they state, which Aghabozorgi et al. (2023) later generalise.

3 METHOD

We fine-tune an autoregressive language model to generate W consecutive tokens of a response in one forward pass. Section 2.1 states that dependence among the tokens of a block needs a mechanism that couples their positions. The block-diffusion family couples them across its denoising steps (Arriola et al., 2025). We couple them within a single forward pass, through one latent vector $\mathbf{z}$ drawn once for those W positions and supplied to the model at each of them. The construction adds one projection matrix to a pretrained decoder-only transformer, and the latent enters the model through that matrix. The construction also limits what each of those W positions attends. Nothing else is added, no other parameter, no extra head, no second model and no new token, and the rest

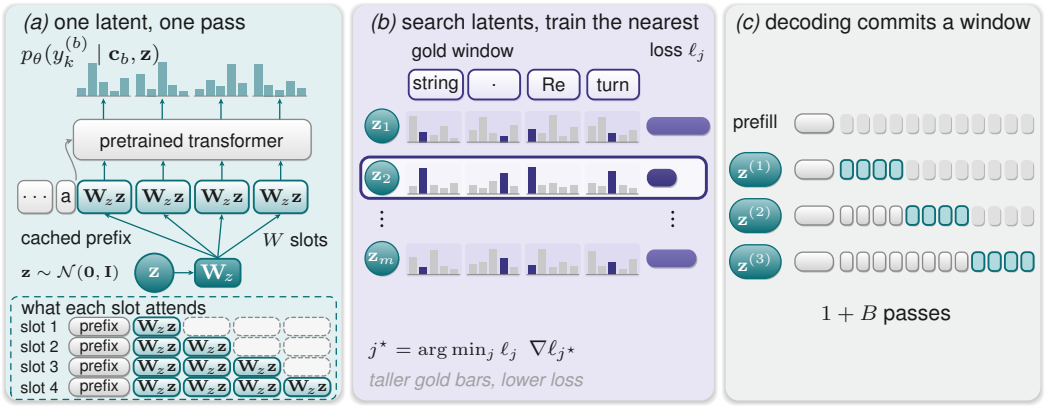


Figure 2: The method generates a window of W response tokens in one pass. *(a)* The projected latent $\mathbf{W}_z\mathbf{z}$ is the input of all W slots appended to the prefix $\mathbf{c}_b$. *(b)* Training scores the ground-truth window under each of m latents, one loss bar each, and takes the gradient on the nearest draw $j^\star$ alone. *(c)* Decoding reads the prompt in one pass, then draws one latent per window and commits the window, so B windows cost $1 + B$ passes.

of the architecture is unchanged. Every decoder-only transformer takes a vector and a position for each input and applies an attention mask, so the construction applies to any pretrained model of that kind. Section 3.1 defines the model, Section 3.2 the training objective, conditional IMLE applied to windows, and Section 3.3 the decoding loop, one forward pass per window (Figure 2).

3.1 Latent-Conditioned Windows

A window is at most W consecutive tokens of a response. The model decodes a response as a sequence of successive windows, which never overlap. Those windows are the blocks of Equation 1 (Wang et al., 2018; Arriola et al., 2025). The model conditions a window on its prefix and on a latent. For window b the prefix $\mathbf{c}_b$ is the prompt together with every response token before the window. For a decoded window the prefix is the $(\mathbf{c}, \mathbf{y}^{(<b)})$ of Equation 1. We draw a latent $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ for the window, and we call one such latent a draw. We append W inputs after the prefix and call them the window's slots (Figure 2a). Slot k sits at the position of token k of the window. The projection matrix of the construction is $\mathbf{W}_z \in \mathbb{R}^{d \times d_z}$, with $d_z = 128$ the latent's dimension and d the model's hidden size. Every slot carries the same vector $\mathbf{W}_z\mathbf{z}$ in place of a token embedding. Slot k attends only its window's prefix, itself and the $k - 1$ earlier slots of its window. In training, where many windows share one sequence, a custom attention mask enforces this view (Appendix A.1). At decode time the ordinary causal mask gives the same view, because the slots come last in each forward pass and no slot of an earlier window is kept for later passes (Section 3.3).

One forward pass over the W slots, on top of a prefix the model has already encoded, produces the model's distribution over the vocabulary at every slot,

$$\begin{aligned} \text{input of slot } k &= \mathbf{W}_z\mathbf{z} \text{ at the position of } y_k^{(b)}, \\ \text{output of slot } k &= p_\theta\big(y_k^{(b)} \mid \mathbf{c}_b, \mathbf{z}\big), \qquad k = 1, \ldots, W. \end{aligned} \tag{3}$$

Because every slot carries the same vector $\mathbf{W}_z\mathbf{z}$ and attends neither a token of its own window nor a slot of another window, the output at slot k depends only on the prefix, on $\mathbf{z}$ and on the slot's position. The W slot distributions are all the model computes for a window, and we take their product as the window's conditional $p_\theta(\mathbf{y}^{(b)} \mid \mathbf{c}_b, \mathbf{z})$. Given the prefix and $\mathbf{z}$, the W tokens of a window are therefore independent. The block factor $p_\theta(\mathbf{y}^{(b)} \mid \mathbf{c}_b)$ of Equation 1 is the expectation of the window's conditional over $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. An expectation of a product need not be a product, so under the block factor the tokens of a window can depend on one another. That dependence comes from the latent alone, the one mechanism in the construction that couples the positions of a window. IMLE-LLM, our name for a model fine-tuned with this construction, is an implicit model in the sense of Section 2.2. To sample a window from it, we draw a latent and then sample one token from each slot's distribution. Windows sampled this way follow the block factor, which has no closed form. We train it with conditional IMLE, an estimator for implicit models that draws every latent from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, so training needs no network that infers a latent from a ground-truth window.

3.2 Training Objective

We instantiate Equation 2 for each training window. Training data are prompt and response pairs. From each response we form training windows at stride s, the number of positions between the start of one window and the start of the next. The runs of this paper use $s = 1$, so a response of $n \geq W$ tokens gives $n - W + 1$ windows. Consecutive windows overlap in $W - 1$ positions, and each response token is the target of slot k in up to W windows, with a different k in each. The input $\mathbf{x}$ of Equation 2 is the prefix $\mathbf{c}_b$ of training window b. The response tokens of that prefix are ground truth, except where the scheduled-sampling rule below replaces them. For each window b we draw m latents $\mathbf{z}_{b,1}, \ldots, \mathbf{z}_{b,m}$, with $m = 4W$ as our convention. We read $T_\theta(\mathbf{c}_b, \mathbf{z}_{b,j})$ as the window's conditional under draw j. The distance $\mathcal{L}$ from the ground-truth window to that conditional is built from the negative log-likelihood of each ground-truth token under the conditional, so scoring a draw generates no token. Without a weight, a response token would enter the objective once for every training window that covers it. We therefore weight the negative log-likelihood of response position t by $1/\kappa_t$, where κ_t is the number of training windows that cover that position. The window loss $\ell_j^{(b)}$ of draw j sums those weighted terms over the slots of window b that fall inside the response, where token $y_k^{(b)}$ is the response token y_t,

$$\mathcal{L}\big(T_\theta(\mathbf{c}_b, \mathbf{z}_{b,j}), \mathbf{y}^{(b)}\big) \;=\; \ell_j^{(b)} \;=\; \sum_{\substack{1 \leq k \leq W \\ (b-1)s+k \leq n}} \frac{1}{\kappa_t} \Big(-\log p_\theta\big(y_t \mid \mathbf{c}_b, \mathbf{z}_{b,j}\big) \Big), \qquad t = (b-1)s + k. \tag{4}$$

At $s = W$ every κ_t is one and the window loss is an unweighted sum.

We score the window under each of its m draws without gradient. The window's nearest draw is $j^\star = \arg\min_j \ell_j^{(b)}$, and we select that draw (Figure 2b). An update step runs over a batch of examples. The objective is the sum of $\ell_{j^\star}^{(b)}$ over the windows of that batch, divided by T, the number of response tokens in it,

$$\frac{1}{T} \sum_b \min_{j \in \{1, \ldots, m\}} \ell_j^{(b)}. \tag{5}$$

No other loss term enters the objective. Because the weights of each response token sum to one, the objective is an average over the response tokens of the batch. We take the gradient on each window's selected draw alone. As in the procedure of Li et al. (2020), we select each window's nearest draw once for a group of examples, then take several update steps on batches from that group before selecting again. We call one such cycle a round. Appendix A.1 gives the settings of a round and the full training step (Algorithm 1). Equation 5 is a sampled form of the expectation in Equation 2, over the windows of one batch and with the m latents drawn afresh every round. Scoring each window's draws against its own ground-truth tokens is the per-input matching rule of Section 2.2. Because the nearest draw of every window enters the objective, no training window is left without a draw pulled toward it.

A model trained only on ground-truth tokens meets a different prefix when it decodes, a gap Ranzato et al. (2016) call exposure bias. Scheduled sampling (Bengio et al., 2015) addresses that gap by feeding a model its own tokens part of the time. Our scheduled-sampling rate ρ is the probability, per training example and per round, that the model's own tokens replace the ground-truth tokens of one window in the prefix of later windows (Appendix A.1). The targets of every window stay ground truth. The original method raises this probability as training proceeds. We hold ρ fixed (Appendix B.2).

We update every transformer layer of the backbone during training with BAdam (Luo et al., 2024), one layer at a time. The input embeddings, the output head and the final layer norm stay fixed, and $\mathbf{W}_z$ trains at every step. For the first 792 update steps only the projection matrix trains and the backbone stays frozen, as in Liu et al. (2023a), because moving the backbone before new parameters are fitted distorts pretrained features (Kumar et al., 2022).

3.3 Decoding

We decode a response one window at a time, in the order of the blocks of Equation 1. The window size at decode time is the one the model was trained with. Decoding advances by W positions rather

than by the stride $s = 1$ of training, so each decoded window starts where the previous one ended (Figure 2c). Were a decoded window to overlap the previous one, part of its forward pass would be spent on positions already decoded. The decoding loop runs one forward pass per window on top of a key-value cache, which stores the keys and values of the positions the model has already read (Algorithm 2 in Appendix A.2). One forward pass over the prompt, the prefill, fills that cache before the first window. For each window we then draw a fresh latent $\mathbf{z}$, and nothing else in the loop is random. Each window's forward pass runs over the tokens of the previous window, if there is one, followed by its own W slots. We take the most likely token at each slot and discard the slots' cache entries, because they are computed from the latent and sit at the positions the committed tokens must take. We then commit those tokens to the response in order, up to the first stop token, and never change them afterwards. The next pass encodes the committed tokens at those positions with no latent in view, as the prefix is encoded in training (Appendix A.1). Generation ends at a stop token, which is never emitted, at a stop string found in the decoded text, or at the token budget.

Two steps of the loop, the single draw per window and the most likely token at each slot, rest on the sampler of Section 2.2 and on the model of Section 3.1. That sampler takes one latent for one input (Li et al., 2020), here the window's prefix. The m draws and the nearest-draw selection belong to the training objective, which scores each draw against a ground-truth window that decoding does not have. We therefore commit a window from a single draw, with no selection among draws. Taking the most likely token at each slot is greedy decoding of the draw's conditional, with no expectation over $\mathbf{z}$. Because the W tokens are independent given the prefix and the draw (Section 3.1), that choice gives the most likely window under the draw. Across draws, the tokens of a decoded window can still depend on one another, since all W of them are decoded from the same draw.

A response of B windows costs $1 + B$ forward passes, the prefill and one pass per window. The model commits W tokens in every window before the last, so a response of n tokens uses about n/W windows, and the forward count falls as W grows. The fine-tuned model is the only model in the loop, with no draft model and no verifier. Section 4 times our decoding against a one-token-at-a-time reference, the same fine-tuned weights with no slot and no latent, under the protocol of Appendix D, and reports the accuracy of both. A block model's samples can differ from an autoregressive model's (Section 2.1), and no term of our objective makes them agree. Section 4 therefore also measures accuracy against an autoregressive control, a separate fine-tune from the same initialisation and the same data.

4 Experiments

We measure the accuracy IMLE-LLM keeps and the speed-up it reaches at $W = 4$, against the model it starts from, an autoregressive control and six public baselines. From here on, IMLE-LLM names this $W = 4$ model. The same recipe at $W = 2$, 3 and 5 gives the sweep models, which serve only for comparison and show how accuracy trades against speed-up.

4.1 Experimental Setup

IMLE-LLM and the sweep models fine-tune Qwen3-8B (Yang et al., 2025) in its non-thinking mode with the conditional IMLE objective of Section 3.2, $m = 4W$ latent draws per window and a scheduled-sampling rate $\rho = 0.25$ (Appendix B.2). All our runs train on one corpus of public math, code, and general instruction and chat prompts. Qwen3-8B itself writes every response (Appendix B.1). All our runs start from the original Qwen3-8B. The AR fine-tuned model is the autoregressive (AR) control of Section 3.3, trained as our runs are but with the next-token loss.

Baselines. We compare with DFlash (Chen et al., 2026), a lossless speculative decoder on Qwen3-8B; WeDLM-8B (Liu et al., 2026a), ReFusion (Li et al., 2026) and SDAR-8B (Cheng et al., 2026), diffusion decoders built on Qwen3-8B; Nemotron-Labs-Diffusion 8B (Nemotron-LD; Fu et al., 2026), a diffusion decoder built on Ministral3-8B; and Kirchenbauer et al. (2026), a multi-token predictor built on Llama-3.1-8B. Each runs its authors' released code, weights and settings under our protocol, on the same GPU as our runs (Appendix D).

Evaluation. We score six benchmarks by generation, GSM8K (Cobbe et al., 2021) and MATH-500 (Hendrycks et al., 2021b; Lightman et al., 2024) for math, HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021b; Liu et al., 2023b) and LiveCodeBench v6 (Jain et al., 2025) for code,

Table 1: Accuracy and speed-up of IMLE-LLM ($W = 4$), the sweep models and baselines (higher is better). $\times$ is tokens per second over each system's own base decoded token by token (one L40S, batch 1). WeDLM, ReFusion and SDAR share our Qwen3-8B base, and IMLE-LLM is more accurate and faster than all three. DFlash is lossless, so its accuracy (dashes) equals the original row. Nemotron-LD's base is its own autoregressive mode, and the Llama-3.1-8B Magpie SFT row is the base of Kirchenbauer et al. (2026), not a baseline. Rows decoded one token at a time are references, so their speed-up cells are dashes. LCB is LiveCodeBench and TPF is tokens per forward pass.

	Accuracy (%)								Speed-up vs. each system's own base model							
	Math		Code			Instr.	Knowl.	Comm.	Math		Code			Instr.		
Model	GSM8K	MATH-500	HumanEval	MBPP	LCB	IFEval	MMLU	HellaSwag	GSM8K	MATH-500	HumanEval	MBPP	LCB	IFEval	Mean	TPF
References																
original Qwen3-8B, one token	93.48	85.00	82.32	82.54	28.19	82.07	75.00	75.10	—	—	—	—	—	—	—	1
AR fine-tuned Qwen3-8B, one token	92.27	83.20	80.49	81.75	27.09	80.41	75.22	76.68	—	—	—	—	—	—	—	1
Lossless speculative decoding																
DFlash (Qwen3-8B)	—	—	—	—	—	—	—	—	3.96×	4.67×	3.95×	3.57×	4.23×	1.45×	3.64×	4.52
Lossy, built on Qwen3-8B																
WeDLM-8B (Qwen3-8B)	89.08	77.60	74.39	76.98	18.72	66.91	75.04	76.16	2.56×	2.63×	3.26×	2.94×	2.98×	1.97×	2.72×	3.31
ReFusion (Qwen3-8B)	84.91	76.40	76.22	76.19	19.60	65.62	73.70	75.82	1.94×	2.07×	3.14×	2.82×	2.35×	1.77×	2.35×	2.65
SDAR-8B (Qwen3-8B)	87.19	76.00	75.00	75.93	16.30	63.59	74.41	76.20	1.91×	2.13×	1.85×	1.83×	1.55×	1.32×	1.77×	2.14
Lossy, other 8B base models																
Nemotron-LD 8B	93.25	86.80	79.88	84.13	29.74	70.43	79.91	76.08	1.77×	1.86×	1.79×	1.41×	1.33×	1.88×	1.67×	1.86
Llama-3.1-8B Magpie SFT	69.52	31.00	43.29	40.74	13.88	55.08	67.92	79.12	—	—	—	—	—	—	—	1
Kirchenbauer et al. (2026)	64.06	28.40	37.80	34.66	11.23	40.30	65.13	76.83	2.71×	2.89×	2.13×	2.08×	2.12×	2.51×	2.41×	3.03
Ours, one model writing W tokens per pass																
$W = 2$, window	91.23	82.07	78.66	80.42	26.36	76.03	75.30	76.53	1.63×	1.72×	1.71×	1.72×	1.82×	1.75×	1.73×	1.97
$W = 3$, window	90.93	81.80	78.05	79.89	26.14	74.74	75.28	76.45	2.56×	2.54×	2.61×	2.48×	2.50×	2.63×	2.55×	2.91
$W = 4$, window (our model)	89.89	80.87	76.63	78.48	25.55	71.23	75.26	76.36	3.43×	3.44×	3.44×	3.25×	3.28×	3.51×	3.39×	3.88
$W = 5$, window	86.35	77.53	72.15	74.16	23.49	59.95	75.15	76.14	4.07×	4.18×	4.14×	3.89×	4.04×	4.24×	4.09×	4.70

and IFEval (Zhou et al., 2023) for instruction following. All six follow the protocol of Nemotron-LD in NeMo-Skills, zero-shot, with greedy decoding, non-thinking mode and at most 4,096 new tokens (Appendix C.1). Accuracy is the percentage of problems solved, with IFEval at prompt-level strict accuracy. We also score MMLU (Hendrycks et al., 2021a) and HellaSwag (Zellers et al., 2019) by likelihood, which tests the weights alone. Each generated-benchmark cell of a W row is the mean of three seeds (Appendix C.2). We time every row with a speed-up on one NVIDIA L40S at batch 1 in bf16, with the prefill inside the timed region (Appendix D). The speed-up ($\times$) divides a row's tokens per second by that of the system's own base model decoded one token at a time on the same prompts, the ratio that TwoTower (Reda et al., 2026), Fast-dLLM v2 (Wu et al., 2026) and WeDLM report.

4.2 Main Results

IMLE-LLM reaches a mean speed-up of 3.39$\times$ over the original Qwen3-8B decoded one token at a time, and commits 3.88 tokens per forward pass on average (Table 1). The fair reference for its accuracy is the AR fine-tuned model. Against it, IMLE-LLM keeps 97.3 % of the math accuracy and 95.2 % of the code accuracy (Figure 3). Against its own weights decoded one token at a time (Section 3.3), the speed-up is the same at batch 1 and 2.86$\times$ at batch 16 (Table 6).

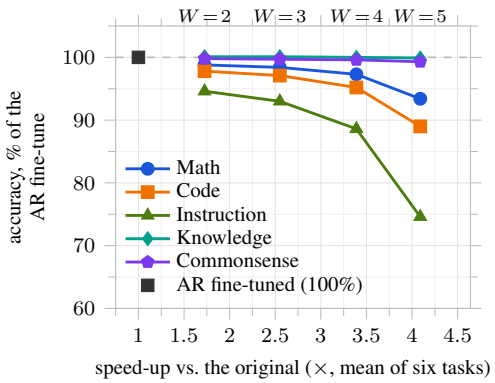


Figure 3: Accuracy per task type, in % of the AR fine-tuned model, against the mean speed-up, for $W = 2$ to 5 (higher is better). A task type averages its benchmarks.

Wider windows trade accuracy for speed (Figure 3). The mean speed-up rises from 1.73$\times$ at $W = 2$ to 4.09$\times$ at $W = 5$ (Table 1). A window pass costs only a little more than a one-token pass, so the speed-up comes from the tokens each pass commits (Appendix E.2). Up to $W = 4$, math and code keep at least 95.2 % of the AR fine-tuned model's accuracy, and at $W = 5$ they keep at least 89.0 %. MMLU and HellaSwag, scored by likelihood, stay within 1 % of the AR fine-tuned model at every W (Figure 3).

In Table 1, IMLE-LLM leads WeDLM, ReFusion and SDAR, the lossy baselines built on the same Qwen3-8B, in both accuracy and speed-up on every generated benchmark (speed-up in Figure 7).

Nemotron-LD and Kirchenbauer et al. (2026) start from other base models, so their rows are read against their own bases.

4.3 Ablation Study

Table 2 changes one part of the recipe at a time.

Table 2: Ablations of the recipe (accuracy in %, higher is better).

Row	Math GSM8K	Math MATH-500	Code HumanEval	Mean
AR fine-tuned	92.27	83.20	80.49	85.32
A. Training at $W = 1$				
Ours, $W = 1$	92.19	83.13	80.49	85.27
D. Same weights, one token at a time				
$W = 2$	92.19	83.20	80.49	85.29
$W = 3$	92.12	83.20	80.49	85.27
$W = 4$	92.12	83.00	80.49	85.20
$W = 5$	92.12	83.00	80.49	85.20
B. Change from $\rho = 0.25$ to 0.5, in points				
$W = 2$	-0.10	-0.14	0.00	-0.08
$W = 3$	-0.18	$+0.07$	-0.20	-0.10
$W = 4$	-0.18	-0.14	$+0.20$	-0.04
$W = 5$	-0.25	-0.33	-0.40	-0.33
C. Draws per window at $W = 2$				
$m = 8$ (the recipe)	91.23	82.07	78.66	83.99
$m = 1$	82.11	74.53	67.28	74.64
$m = 16$	91.41	82.33	78.66	84.13
$m = 32$	91.51	82.40	78.66	84.19

Training and decoding. Ablations A and D separate what training does to accuracy from what decoding in windows does. Trained at $W = 1$, our objective with scheduled sampling matches the AR fine-tuned model on the three-task mean (ablation A). Decoded one token at a time, each of our models from $W = 2$ to 5 matches the AR fine-tuned model as well (ablation D). The difference between our models and the AR fine-tuned model on the three-task mean (Table 1) therefore comes mostly from decoding in windows rather than from training.

Latent draws per window. Ablation C asks whether the objective needs several latent draws m per window, at $W = 2$. With one draw per window ($m = 1$), the three-task mean falls by 9.35 points (Table 2). At $m = 1$ the draw is independent of the ground-truth window. The expected window loss of Equation 4 is then lowest when each slot ignores the draw and predicts its token from the prefix alone, which keeps only the marginal distribution of each token (Section 1). Raising m from the recipe's 8 to 16 or 32 brings only a small gain, so $m = 8$ is already enough. Appendix E.1 shows that the trained model uses its latent (Table 4).

Scheduled-sampling rate. Ablation B compares our two scheduled-sampling rates, $\rho = 0.25$ and 0.5, at every W. At each W the two rates give nearly the same three-task mean, with the lower rate slightly ahead (Table 2). The recipe therefore does not hinge on the choice between them.

5 Related Work

Parallel and blockwise language generation. Methods that generate several tokens per forward pass either propose tokens and check them against an autoregressive model, or learn a blockwise distribution whose samples need not match token-by-token decoding. Speculative decoding checks a drafter's tokens in one parallel pass of the target model, and its acceptance rule keeps the target's output distribution (Leviathan et al., 2023; Chen et al., 2023). Drafts come from a small draft model that reads the target's features (Li et al., 2024; 2025; Chen et al., 2026), extra output heads and prediction modules on the target model (Stern et al., 2018; Cai et al., 2024; Gloeckle et al., 2024; DeepSeek-AI, 2024), or the target model itself at mask positions appended after the prefix (Goel et al., 2026; Liu et al., 2026b; Fu et al., 2026). Jacobi decoding updates all positions of a block in parallel until they match sequential greedy decoding, and lookahead decoding verifies n-grams collected along these iterations (Santilli et al., 2023; Fu et al., 2024). Consistency training fine-tunes a pretrained model to map any intermediate guess of this iteration to its fixed point (Kou et al., 2024; Hu et al., 2026). Non-autoregressive translation models and discrete diffusion language models instead learn a blockwise distribution and generate all output positions in parallel, in one pass (Gu et al., 2018) or over refinement or denoising steps (Ghazvininejad et al., 2019; Austin et al., 2021a; Lou et al., 2024; Sahoo et al., 2024; Nie et al., 2025). Semi-autoregressive translation and block diffusion add an autoregressive order over blocks of tokens and fill each block in parallel, in one pass or by denoising inside it (Wang et al., 2018; Han et al., 2023; Arriola et al., 2025). Recent work converts pretrained autoregressive models into diffusion language models that denoise the whole sequence (Gong et al., 2025; Ye et al., 2025) or fill mask positions after the committed text over one or more denoising steps (Cheng et al., 2026; Wu et al., 2026; Reda et al., 2026; Fu et al., 2026; Liu et al., 2026a). A pretrained autoregressive model can also be fine-tuned to predict several positions per pass and commit them with no verifier, as in Kirchenbauer et al. (2026), Set Block Decoding

(Gat et al., 2025), an any-order groupwise autoregressive model (Du et al., 2026) and Tang et al. (2026). Our method shares this setting. These models and the converted diffusion models place a mask token, a learned query vector or one noise token per future token at the positions still to be generated. Our slots instead carry the projection of one Gaussian latent drawn per window.

Coupling tokens generated together. A decoder that gives the tokens of one pass a product of per-position distributions drops their dependence, whatever its parameters or training (Gu et al., 2018; Huang et al., 2022). Draft-and-verify methods get this dependence from the target model, which checks each drafted token given the tokens before it (Leviathan et al., 2023; Chen et al., 2023; Stern et al., 2018; Liu et al., 2026b). Diffusion and iterative refinement predict the positions of a sequence or block independently given its current state at each step, so the positions depend on one another across steps rather than within a step (Austin et al., 2021a; Sahoo et al., 2024; Ghazvininejad et al., 2019; Arriola et al., 2025). Copula and energy-based diffusion models add dependence within a denoising step through a second model (Liu et al., 2025a; Xu et al., 2025). Latent-variable models make the tokens of one pass independent given a latent, so their dependence comes from the expectation over it (Kaiser et al., 2018; Ma et al., 2019; Shu et al., 2020; Peng et al., 2026; Xie et al., 2026; Hayakawa et al., 2025; Grivas et al., 2026). Their latent is a sequence in parallel translation, a Gaussian vector or a random scalar drawn at each step in masked diffusion, and a set of discrete variables in a drafter whose drafts a verifier checks. Those with a sequence or a Gaussian vector learn an encoder or approximate posterior that gives each training example its latent. Our method learns no encoder or posterior. It draws every latent from a fixed Gaussian prior, in training as in decoding, so no network infers a latent from a ground-truth window. Other models make the tokens of one pass a function of random noise, which carries their dependence (Tang et al., 2026; Draxler et al., 2026; Zhang et al., 2026; Liu et al., 2025b;c; Yao, 2026; Shao et al., 2025). The noise is drawn once per future token and fed as input, or it is added as Gaussian noise to mask embeddings, or it enters an output head whose vector an autoencoder decodes into a chunk of tokens. Our latent is one Gaussian vector drawn once per window and supplied as the same input at every slot.

Implicit maximum likelihood estimation. IMLE is a likelihood-free estimator for implicit models, networks that map one Gaussian draw to a sample in one forward pass (Li & Malik, 2018). Later variants change how IMLE builds or draws its latent codes in training, or adapt its objective to each data point (Peng et al., 2022; Vashist et al., 2024; Aghabozorgi et al., 2023). Conditional IMLE has been applied to super-resolution and to image synthesis from semantic layouts (Li et al., 2018; 2019; 2020). Recent work trains one-step generators of robot actions, planned trajectories and human trajectory forecasts with conditional IMLE or its variants in place of a multi-step denoising or flow model, in one case by fine-tuning the action head of a pretrained vision-language-action model (Rana et al., 2025; Lee et al., 2026; Fu et al., 2025; Hosseinkhani et al., 2026). Our method uses conditional IMLE to fine-tune a pretrained autoregressive language model into an implicit model of each window of text, with the window's prefix as the input. To our knowledge, no prior work fine-tunes a pretrained autoregressive language model in this way to commit each window from one forward pass without a verifier, with any dependence among the window's tokens coming from one Gaussian latent drawn once per window and supplied as the same input at every slot.

6 Discussion and Conclusion

Limitations and future work. By design, IMLE-LLM uses no drafter or check, so its output can differ from the original model's. The price is some accuracy, more at wider windows, and narrowing this gap is future work. Each model also serves one window size, and training one model for several sizes would let users choose the size per task. We use one backbone in its non-thinking mode, and larger models, other backbones and the thinking mode remain to be tested. Finally, reinforcement learning could extend our supervised training, with the latent's random draws supplying varied answers.

Conclusion. Tokens written in the same forward pass cannot see one another, and most methods restore the missing dependence outside the pass. IMLE-LLM keeps it inside. One Gaussian draw, shared by every position of a window, carries this dependence. It makes a pretrained language model an implicit model of its next window, which conditional IMLE fits without an extra network. Our model reaches more than a threefold speed-up while keeping most of the accuracy of an autoregressive fine-tune.

AI USE STATEMENT

We used large language models to check grammar and spelling and to suggest clearer wording for text we wrote. We also used them to help find related work, and we read every cited paper ourselves. They helped us write and debug code, check data-processing scripts and write the code that draws figures, and we tested this code ourselves. We asked them for general feedback on the paper. They did not produce the research ideas, the method or the results. We checked all AI-assisted text and code ourselves, and we take full responsibility for the content of this paper.

ETHICS STATEMENT

This work follows the ICLR Code of Ethics. It fine-tunes a large language model to write several tokens in each forward pass, so that the model decodes faster. It involves no human subjects and collects no new data from people. The training prompts come from public datasets, and Qwen3-8B, the model we start from, writes every training response (Appendix B.1). As with other work on faster decoding, the method could also lower the cost of producing harmful text.

REPRODUCIBILITY STATEMENT

Section 3 describes the method, and Appendix A gives its pseudocode in Algorithms 1 and 2. Appendix B gives the training corpus, the settings of every run and the compute. Appendix C gives the evaluation protocol, the graders, the seeds and the repeats. Appendix D gives the speed measurement protocol. Section 4.1 states how each baseline is run. Code will be released upon acceptance.

REFERENCES

Mehran Aghabozorgi, Shichong Peng, and Ke Li. Adaptive IMLE for few-shot pretraining-free generative modelling. In *International Conference on Machine Learning (ICML)*, volume 202 of *Proceedings of Machine Learning Research*, pp. 248–264. PMLR, 2023. URL https://proceedings.mlr.press/v202/aghabozorgi23a.html.

Marianne Arriola, Aaron Kerem Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block Diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations (ICLR)*, pp. 50726–50753, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/7ede97c3e082c6df10a8d6103a2eebd2-Abstract-Conference.html.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, volume 34, pp. 17981–17993, 2021a. URL https://papers.nips.cc/paper_files/paper/2021/hash/958c530554f78bcd8e97125b70e6973d-Abstract.html.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021b. URL https://arxiv.org/abs/2108.07732.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 28, pp. 1171–1179, 2015.

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 *International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235. PMLR, 2024. URL https://proceedings.mlr.press/v235/cai24b.html.

Sahil Chaudhary. Code alpaca: An instruction-following LLaMA model for code generation. GitHub repository, 2023. URL https://github.com/sahil280114/codealpaca.

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.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=Oz335dV48X.

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.

Shuang Cheng, Yihan Bian, Dawei Liu, Yuhua Jiang, Yihao Liu, Linfeng Zhang, Qian Yao, Zhongbo Tian, Wenhai Wang, Qipeng Guo, Kai Chen, Biqing Qi, and Bowen Zhou. SDAR: A synergistic diffusion-AutoRegression paradigm for scalable sequence generation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 22058–22075, San Diego, California, United States, 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.findings-acl.1110. URL https://aclanthology.org/2026.findings-acl.1110/.

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. DeepSeek-V3 technical report, 2024. URL https://arxiv.org/abs/2412.19437.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3029–3051, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.183.

Felix Draxler, Justus Will, Farrin Marouf Sofian, Theofanis Karaletsos, Sameer Singh, and Stephan Mandt. Parallel token prediction for language models. In *International Conference on Learning Representations (ICLR)*, pp. 5624–5652, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/0a43c96ee43ce2e3d05d393c6426c284-Abstract-Conference.html.

Tianqi Du, Lizhe Fang, Weijie Yang, Chenheng Zhang, Zeming Wei, Yifei Wang, and Yisen Wang. Autoregressive models rival diffusion models at ANY-ORDER generation. In *International Conference on Learning Representations (ICLR)*, pp. 88647–88663, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/8ee95cb7a0bbc000beb0c906e02487ef-Abstract-Conference.html.

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

Yonggan Fu, Lexington Whalen, Abhinav Garg, Chengyue Wu, Maksim Khadkevich, Nicolai Oswald, Enze Xie, Daniel Egert, Sharath Turuvekere Sreenivas, Shizhe Diao, Chenhan Yu, Ye Yu, Weijia Chen, Sajad Norouzi, Jingyu Liu, Shiyi Lan, Ligeng Zhu, Jin Wang, Jindong Jiang, Morteza Mardani, Mehran Maghoumi, Song Han, Ante Jukić, Nima Tajbakhsh, Jan Kautz,

and Pavlo Molchanov. Nemotron-Labs-Diffusion: A tri-mode language model unifying autoregressive, diffusion, and self-speculation decoding, 2026. URL https://arxiv.org/abs/2607.05722.

Yuxiang Fu, Qi Yan, Lele Wang, Ke Li, and Renjie Liao. MoFlow: One-step flow matching for human trajectory forecasting via implicit maximum likelihood estimation based distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17282–17293, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Fu_MoFlow_One-Step_Flow_Matching_for_Human_Trajectory_Forecasting_via_Implicit_CVPR_2025_paper.html.

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

Itai Gat, Heli Ben-Hamu, Marton Havasi, Daniel Haziza, Jeremy Reizenstein, Gabriel Synnaeve, David Lopez-Paz, Brian Karrer, and Yaron Lipman. Set block decoding is a language model inference accelerator, 2025. URL https://arxiv.org/abs/2509.04185.

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, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1633. URL https://aclanthology.org/D19-1633/.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Rozière, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15706–15734. PMLR, 2024. URL https://proceedings.mlr.press/v235/gloeckle24a.html.

Raghavv Goel, Mukul Gagrani, Mingu Lee, and Chris Lott. Efficient training-free multi-token prediction via embedding-space probing. In *International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=mKj4fO0BU4.

Shansan Gong, Shivam Agarwal, Yizhe Zhang, Jiacheng Ye, Lin Zheng, Mukai Li, Chenxin An, Peilin Zhao, Wei Bi, Hao Peng, Jiawei Han, and Lingpeng Kong. Scaling diffusion language models via adaptation from autoregressive models. In *International Conference on Learning Representations (ICLR)*, pp. 5046–5073, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/0fa81c3f0d57f95b8776de3a248ef0ed-Abstract-Conference.html.

Andreas Grivas, Lorenzo Loconte, Emile van Krieken, Piotr Nawrot, Yu Zhao, Euan Wielewski, Pasquale Minervini, Edoardo Ponti, and Antonio Vergari. Fast and expressive multi-byte prediction with probabilistic circuits. In *International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=6kCEyw9god.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. Non-autoregressive neural machine translation. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://arxiv.org/abs/1711.02281.

Xiaochuang Han, Sachin Kumar, and Yulia Tsvetkov. SSD-LM: Semi-autoregressive simplex-based diffusion language model for text generation and modular control. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11575–11596, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.647. URL https://aclanthology.org/2023.acl-long.647/.

Satoshi Hayakawa, Yuhta Takida, Masaaki Imaizumi, Hiromi Wakaki, and Yuki Mitsufuji. Distillation of discrete diffusion through dimensional correlations. In *International Conference*

on Machine Learning (ICML), volume 267 of *Proceedings of Machine Learning Research*, pp. 22259–22297. PMLR, 2025. URL https://proceedings.mlr.press/v267/hayakawa25a.html.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations (ICLR)*, 2021a. URL https://openreview.net/forum?id=d7KBjmI3GmQ.

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. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS)*, 2021b. URL https://arxiv.org/abs/2103.03874.

Kian Hosseinkhani, Qinhe Peng, George Shramko, Mehran Aghabozorgi, Jianing Qian, Tristan Engst, Alireza Moazeni, Dinesh Jayaraman, and Ke Li. IMLE-VLA: Fast single-step action generation for vision-language-action policies. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2026. URL https://arxiv.org/abs/2609.10915. To appear.

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 *International Conference on Machine Learning (ICML)*, 2026. URL https://openreview.net/forum?id=ORy2hyPd4G.

Fei Huang, Tianhua Tao, Hao Zhou, Lei Li, and Minlie Huang. On the learning of non-autoregressive transformers. In *International Conference on Machine Learning (ICML)*, volume 162 of *Proceedings of Machine Learning Research*, pp. 9356–9376. PMLR, 2022. URL https://proceedings.mlr.press/v162/huang22k.html.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida I. Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=chfJJYC3iL.

Łukasz Kaiser, Aurko Roy, Ashish Vaswani, Niki Parmar, Samy Bengio, Jakob Uszkoreit, and Noam Shazeer. Fast decoding in sequence models using discrete latent variables. In *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2390–2399. PMLR, 2018. URL https://proceedings.mlr.press/v80/kaiser18a.html.

John Kirchenbauer, Abhimanyu Hans, Brian Bartoldson, Micah Goldblum, Ashwinee Panda, and Tom Goldstein. Multi-token prediction via self-distillation. In *AdaptFM: Resource-Adaptive Foundation Model Inference, Workshop at the International Conference on Machine Learning (ICML)*, 2026. URL https://arxiv.org/abs/2602.06019.

Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. CLLMs: Consistency large language models. In *International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 25426–25440. PMLR, 2024. URL https://proceedings.mlr.press/v235/kou24a.html.

Ananya Kumar, Aditi Raghunathan, Robbie Jones, Tengyu Ma, and Percy Liang. Fine-tuning can distort pretrained features and underperform out-of-distribution. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://openreview.net/forum?id=UYneFzXSJWh.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. 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 (SOSP '23)*, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Grayson Lee, Minh Bui, Shuzi Zhou, Yankai Li, Mo Chen, and Ke Li. Implicit maximum likelihood estimation for real-time generative model predictive control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026. URL https://arxiv.org/abs/2603.13733.

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

Jia-Nan Li, Jian Guan, Wei Wu, and Chongxuan Li. ReFusion: A diffusion large language model with parallel autoregressive decoding. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=LBtWaUc7FE.

Ke Li and Jitendra Malik. Implicit maximum likelihood estimation, 2018. URL https://arxiv.org/abs/1809.09087.

Ke Li, Shichong Peng, and Jitendra Malik. Super-resolution via conditional implicit maximum likelihood estimation, 2018. URL https://arxiv.org/abs/1810.01406.

Ke Li, Tianhao Zhang, and Jitendra Malik. Diverse image synthesis from semantic layouts via conditional IMLE. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4220–4229, 2019. URL https://openaccess.thecvf.com/content_ICCV_2019/html/Li_Diverse_Image_Synthesis_From_Semantic_Layouts_via_Conditional_IMLE_ICCV_2019_paper.html.

Ke Li, Shichong Peng, Tianhao Zhang, and Jitendra Malik. Multimodal image synthesis with conditional implicit maximum likelihood estimation. *International Journal of Computer Vision*, 128 (10–11):2607–2628, 2020. doi: 10.1007/s11263-020-01325-y.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In *Advances in Neural Information Processing Systems*, volume 38, pp. 136737–136756, 2025. doi: 10.52202/085713-4562. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/c7b5a35ea98b62512a869c19ea7b03cb-Abstract-Conference.html.

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. In *International Conference on Learning Representations (ICLR)*, 2024. URL https://arxiv.org/abs/2305.20050.

Aiwei Liu, Minghua He, Shaoxun Zeng, Sijun Zhang, Linhao Zhang, Chuhan Wu, Wei Jia, Yuan Liu, Xiao Zhou, and Jie Zhou. WeDLM: Reconciling diffusion language models with standard causal attention for fast inference. In *International Conference on Machine Learning (ICML)*, 2026a. URL https://openreview.net/forum?id=QwtmbKAOZU.

Anji Liu, Oliver Broadrick, Mathias Niepert, and Guy Van den Broeck. Discrete copula diffusion. In *International Conference on Learning Representations (ICLR)*, pp. 88953–88979, 2025a. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/dd1fef536655685898a6602bfbf16857-Abstract-Conference.html.

Enshu Liu, Qian Chen, Xuefei Ning, Shengen Yan, Guohao Dai, Zinan Lin, and Yu Wang. Distilled decoding 2: One-step sampling of image auto-regressive models with conditional score distillation. In *Advances in Neural Information Processing Systems*, volume 38, pp. 5448–5476, 2025b. doi: 10.52202/085713-0193. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/085f2a21d3f0cef1585fceaa6dacdc15-Abstract-Conference.html.

Enshu Liu, Xuefei Ning, Yu Wang, and Zinan Lin. Distilled decoding 1: One-step sampling of image auto-regressive models with flow matching. In *International Conference on Learning Representations (ICLR)*, pp. 42150–42173, 2025c. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/67d3805a7b01672c1c2dcb30f6d5290f-Abstract-Conference.html.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Advances in Neural Information Processing Systems*, 2023a.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023b. doi: 10.52202/075280-0943. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html.

Jingyu Liu, Xin Dong, Zhifan Ye, Rishabh Mehta, Yonggan Fu, Vartika Singh, Jan Kautz, Ce Zhang, and Pavlo Molchanov. TiDAR: Think in diffusion, talk in autoregression. In *Proceedings of Machine Learning and Systems*, volume 8, pp. 748–762, 2026b. URL https://proceedings.mlsys.org/paper_files/paper/2026/hash/1367d856028f65a9555b0274db09e608-Abstract-Conference.html.

LMDeploy Contributors. LMDeploy: A toolkit for compressing, deploying, and serving LLM. GitHub repository, 2023. URL https://github.com/InternLM/lmdeploy.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *International Conference on Machine Learning (ICML)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32819–32848. PMLR, 2024. URL https://proceedings.mlr.press/v235/lou24a.html.

Qijun Luo, Hengxu Yu, and Xiao Li. BAdam: A memory efficient full parameter optimization method for large language models. In *Advances in Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=0uXtFk5KNJ.

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, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1437. URL https://aclanthology.org/D19-1437/.

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 *Advances in Neural Information Processing Systems*, volume 38, pp. 50608–50646, 2025. doi: 10.52202/085713-1689. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/48b383b24230e0e6e649d9c98dae4d8c-Abstract-Conference.html.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. Instruction tuning with GPT-4, 2023. URL https://arxiv.org/abs/2304.03277.

Fred Zhangzhi Peng, Avishek Joey Bose, Anru R. Zhang, and Alexander Tong. Coupling models for one-step discrete generation, 2026. URL https://arxiv.org/abs/2605.07193.

Shichong Peng, Alireza Moazeni, and Ke Li. CHIMLE: Conditional hierarchical IMLE for multimodal conditional image synthesis. In *Advances in Neural Information Processing Systems*, volume 35, pp. 280–296, 2022. doi: 10.52202/068431-0021. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/0207c9ea9faf66c6e892c3fa3c167b75-Abstract-Conference.html.

Krishan Rana, Robert Lee, David Pershouse, and Niko Sünderhauf. IMLE Policy: Fast and sample efficient visuomotor policy learning via implicit maximum likelihood estimation. In *Robotics: Science and Systems (RSS) XXI*, 2025. doi: 10.15607/RSS.2025.XXI.158. URL https://roboticsproceedings.org/rss21/p158.html.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2016.

Fitsum Reda, John Kamalu, Roger Waleffe, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-Labs-TwoTower: Diffusion language modeling with pretrained autoregressive context, 2026. URL https://arxiv.org/abs/2606.26493.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 130136–130184, 2024. doi: 10.52202/079017-4135. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/eb0b13cc515724ab8015bc978fdde0ad-Abstract-Conference.html.

Andrea Santilli, Silvio Severino, Emilian Postolache, Valentino Maiorca, Michele Mancusi, Riccardo Marin, and Emanuele Rodolà. Accelerating transformer inference for translation via parallel decoding. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12336–12355, Toronto, Canada, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.689. URL https://aclanthology.org/2023.acl-long.689/.

Chenze Shao, Darren Li, Fandong Meng, and Jie Zhou. Continuous autoregressive language models, 2025. URL https://arxiv.org/abs/2510.27688.

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*, volume 34, pp. 8846–8853, 2020. doi: 10.1609/aaai.v34i05.6413.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems*, volume 31, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/hash/c4127b9194fe8562c64dc0f5bf2c93bc-Abstract.html.

Zhiwei Tang, Yuanyu He, Yizheng Han, Wangbo Zhao, Jiasheng Tang, Fan Wang, and Bohan Zhuang. K-Forcing: Joint next-K-token decoding via push-forward language modeling, 2026. URL https://arxiv.org/abs/2606.10820.

Chirag Vashist, Shichong Peng, and Ke Li. Rejection sampling IMLE: Designing priors for better few-shot image synthesis. In *Computer Vision – ECCV 2024*, Lecture Notes in Computer Science, pp. 441–456. Springer Nature Switzerland, 2024. doi: 10.1007/978-3-031-72664-4_25. URL https://link.springer.com/chapter/10.1007/978-3-031-72664-4_25.

Chunqi Wang, Ji Zhang, and Haiqing Chen. Semi-autoregressive neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 479–488, Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1044. URL https://aclanthology.org/D18-1044/.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dLLM v2: Efficient block-diffusion LLM. In *International Conference on Learning Representations (ICLR)*, pp. 128353–128370, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/d0865cbe51d35ec322f9af9db7806fc7-Abstract-Conference.html.

Tianyu Xie, Shuchen Xue, Zijin Feng, Tianyang Hu, Jiacheng Sun, Zhenguo Li, and Cheng Zhang. Variational autoencoding discrete diffusion with enhanced dimensional correlations modeling. In *International Conference on Learning Representations (ICLR)*, pp. 39109–39132, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/41c700448b16c2de14dab63c817f90ef-Abstract-Conference.html.

Minkai Xu, Tomas Geffner, Karsten Kreis, Weili Nie, Yilun Xu, Jure Leskovec, Stefano Ermon, and Arash Vahdat. Energy-based diffusion language models for text generation. In *International Conference on Learning Representations (ICLR)*, pp. 33769–33789, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/535baca36883a4e0450423b26b150a48-Abstract-Conference.html.

An Yang et al. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zhaorui Yang, Tianyu Pang, Haozhe Feng, Han Wang, Wei Chen, Minfeng Zhu, and Qian Liu. Self-distillation bridges distribution gap in language model fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1028–1043, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.58.

Lin Yao. A ticket from marginals to joints: Coupled-noise distillation for one-step block generation in diffusion language models, 2026. URL https://arxiv.org/abs/2609.06324.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion large language models, 2025. URL https://arxiv.org/abs/2508.15487.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models, 2023. URL https://arxiv.org/abs/2308.01825.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1472.

Chi Zhang, Xixi Hu, Bo Liu, and Qiang Liu. Gumbel distillation for parallel text generation. In *International Conference on Learning Representations (ICLR)*, pp. 153642–153671, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/f94fd2bfa6ed18837739950150dc4750-Abstract-Conference.html.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems 37 (NeurIPS 2024)*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/724be4472168f31ba1c9ac630f15dec8-Abstract-Conference.html.

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 ALGORITHMS

A.1 TRAINING STEP

Training runs in rounds, and a round draws and scores every latent once before it takes any update step. A round takes a group of 48 examples, draws m latents once for every window of every example, scores and selects once over that group, then takes 12 update steps, each on 8 examples drawn afresh from that group. Each of those subsets is the batch that the objective of Section 3.2 runs over. Algorithm 1 gives the training step, with its inputs on the first line, the model, $\mathbf{W}_z$, the window size W, the stride s, the draws m, the scheduled-sampling rate ρ, the group size and the number of update steps per round.

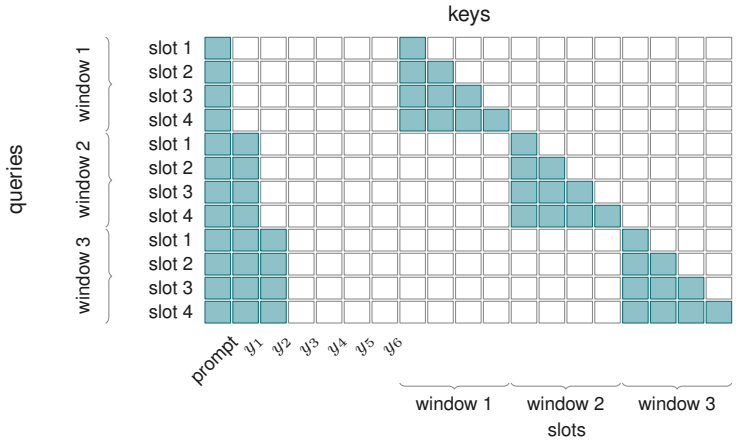


Figure 4: A slot of a training window attends only its own window's prefix, itself and the earlier slots of that window, and the figure shows the whole attention mask cell by cell. Rows are queries, the slots of three overlapping windows, and columns are keys, the prompt, the response tokens and the slots of every window, so a filled cell means that the query of its row attends the key of its column. Each window fills a staircase over its own group of slot columns, leaves the slot columns of the other windows empty, and leaves every token column from its own first token onwards empty. Those empty regions say what the rule forbids, so a slot never sees the tokens of its own window, never sees a token a later window generates, and never sees a slot of another window.

We append the tokenizer's stop token to every response before its windows are formed, so it sits inside the last window as a target. At $s = 1$ no window is padded once the response holds at least W tokens (Figure 5), and a shorter response gives a padded window.

Before we score the draws, we rewrite each example with probability ρ. We draw the boundary of the rewritten window uniformly among all windows but the last. An example with fewer than two windows offers no such boundary, so we never rewrite it. The model at its current weights then generates that window greedily from the ground-truth context with a separate latent, and we take no gradient through that step. Those tokens replace the ground-truth tokens in the context fed to later windows only, and every window's targets stay ground truth.

We then score the draws with two forward passes over the example. The first pass runs over the prompt and the response context, ground truth or rewritten, with a causal mask and no slots, and fills the cache. The second pass runs the W slots of every window of that example on that cache, in chunks of 96 windows. In that pass each slot attends its own window's prefix, itself and the earlier slots of that window, through the attention mask of Section 3.1 (Figure 4).

The loss of a draw is the coverage-weighted sum of Section 3.2, computed in fp32, with weight zero at padding positions. A window's nearest draw is the argmin of that loss over the draws of that window. We then re-run the selected draw of every window of the subset, this time with gradient. We sum the window losses over the subset and divide by its response-token count.

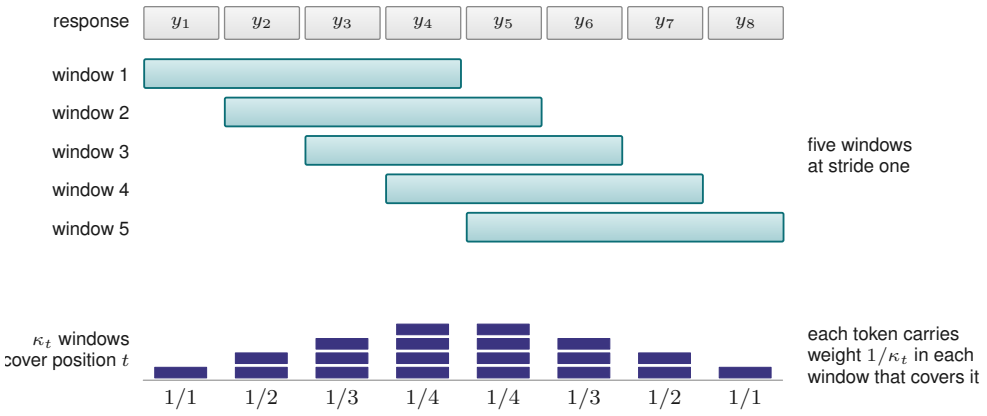


Figure 5: Training forms windows of width W over a response at stride s, which is one in this paper, and the figure shows the overlap that follows. The top strip is the response, and each bar below it is a training window, shifted from the bar above it by the stride. The stack under a response token counts the windows κ_t that cover it, and the fraction beneath the stack is the weight $1/\kappa_t$ that the token carries in each of them. Those weights sum to one over the windows that cover the token.

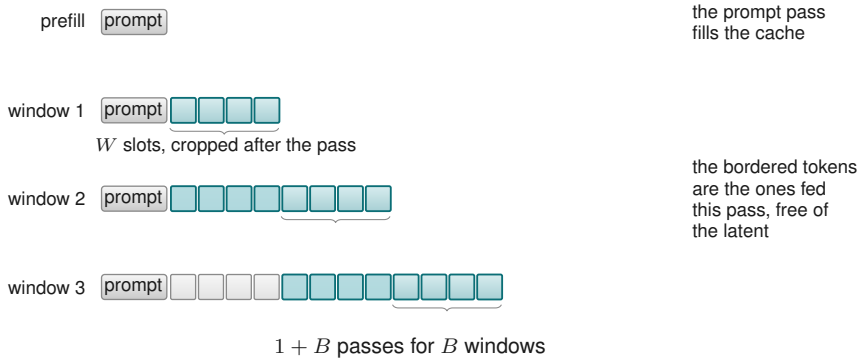


Figure 6: The figure follows what the key-value cache holds and what we feed the model when decoding generates a whole window per forward pass. The first row is the prefill pass over the prompt, and each row below it is the pass of the next window, whose braced cells are that pass's W slots and whose left-hand cells are the tokens committed so far. Only the bordered cells, committed since the previous pass, are fed again beside the slots, and we discard the slots' cache entries after the pass, so the committed tokens enter the cache free of the latent. A response of B windows therefore costs one pass for the prompt and one more for each window.

We train the backbone with BAdam (Luo et al., 2024), with one transformer layer active per update step. We switch the active layer every 22 steps, in ascending order over the 36 layers. The backbone trains at a constant learning rate of 10^{-5}, with no weight decay. $\mathbf{W}_z$ has its own AdamW optimizer, which runs at that same learning rate. Only $\mathbf{W}_z$ moves for the first 792 update steps, because we skip the backbone's update step, and that skip also holds the switch of BAdam's active layer.

The backbone runs in bf16, and we keep $\mathbf{W}_z$ and every loss in fp32. We initialise $\mathbf{W}_z$ from a Gaussian whose standard deviation is the mean row-wise standard deviation of the input-embedding table divided by $\sqrt{d_z}$, so its outputs start at the scale of a token embedding.

Algorithm 1 This box gives the training step in full, with the inputs on the first line and the round structure of Appendix A.1.

Input: model, $\mathbf{W}_z$, window size W, stride s, draws m, scheduled-sampling rate ρ, group size, update steps per round
for each round **do**
  take a group of examples and form windows over every response at stride s
  for each example of the group **do**
    with probability ρ, and only when the example holds two windows or more, pick a boundary uniformly among all windows but the last
    generate that window greedily at the current weights with a separate latent, put its tokens into the context of the later windows, and leave every target ground truth
  end for
  for each example of the group **do**
    run the first pass, with no slots, over the prompt and the response context, which fills the cache
    draw $\mathbf{z}_{b,1}, \ldots, \mathbf{z}_{b,m} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ for every window b of the example
    run the second pass, over the W slots of every window of the example, on that cache, in chunks, without gradient, and score $\ell_j^{(b)}$ for every draw of every window
  end for
  for each window b of the group **do**
    $j^\star \leftarrow \arg\min_j \ell_j^{(b)}$
  end for
  for each update step, on a subset of the group **do**
    re-run the selected draw of every window of the subset, with gradient
    during the warm-up, skip the backbone's step, which also holds the switch of BAdam's active layer
    take an update step on $\frac{1}{T}\sum_b \ell_{j^\star}^{(b)}$, on $\mathbf{W}_z$ and on the backbone's active layer, with T the response-token count of the subset
  end for
end for

A.2 Decoding Loop

Algorithm 2 gives the decoding loop, with its inputs on the first line, the prompt, the window size W, the token budget, the stop tokens and stop strings, and the random seed. One forward pass over the prompt fills the cache, and only the slots' logits are read, so the prompt's last logit is not used.

For a window we feed the model the embeddings of the tokens committed since the previous forward pass, followed by the W slot vectors $\mathbf{W}_z\mathbf{z}$. Those embeddings and slots sit at consecutive positions that continue the cache, and the first window has no committed tokens to feed (Figure 6). One forward pass over that sequence yields the W slot logits. The most likely token at each slot is then taken.

The slots' cache entries hold the latent rather than committed tokens, so we discard them. The window's tokens are then committed left to right, up to the first stop token and up to the remaining budget, and fed with the next pass. We end generation on a stop token, on a stop string found in the decoded text after a commit, or on the budget, in that order of precedence. The number of forward passes is one plus the number of windows used, and we verify that count on every timed run of the loop.

With a one-slot window the same loop runs with $W = 1$. That is still a latent-conditioned window, with a fresh latent per token and one forward pass per token after the prefill. It is not the one-token-at-a-time reference of Section 3.3, which runs the same weights with no slot and no latent.

Algorithm 2 This box gives the decoding loop in full, with the inputs on the first line and the stop rule of Appendix A.2.

Input: prompt, window size W, token budget, stop tokens, stop strings, random seed
run a forward pass over the prompt, which fills the cache
while no stopping condition is met **do**
  draw $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  build the sequence to feed from the tokens committed since the previous pass, then W slots that all carry $\mathbf{W}_z\mathbf{z}$
  run a forward pass over that sequence and read the W slot logits
  take the most likely token at each slot
  discard the slots' entries from the cache, so it holds the prompt and the tokens committed before this window
  commit those tokens left to right, up to the first stop token and up to the remaining budget, to be fed with the next pass
  stop on a stop token, else on a stop string in the decoded text, else on the budget
end while
return the committed tokens, generated in $1 +$ (windows used) forward passes

B Training Details

This appendix gives the corpus and the settings of every run in Section 4.1.

B.1 Training Corpus

The prompts come from the training sets of GSM8K and MATH (Cobbe et al., 2021; Hendrycks et al., 2021b), from CodeAlpaca-20k (Chaudhary, 2023) and Alpaca-GPT4 (Peng et al., 2023), and from 20 thousand first-turn conversations sampled at random from UltraChat-200K (Ding et al., 2023). The corpus holds about 100 thousand prompts in all. Qwen3-8B writes every response with greedy decoding in non-thinking mode, and every example uses its chat template. Training on the model's own responses follows self-distillation fine-tuning (Yang et al., 2024). For GSM8K and MATH we keep only the responses the grader accepts, as in rejection-sampling fine-tuning (Yuan et al., 2023).

B.2 Training Recipe and Compute

Runs. The main runs are IMLE-LLM ($W = 4$) and the sweep models ($W = 2$, 3 and 5) at $\rho = 0.25$, and the same four widths at $\rho = 0.5$, eight runs in all. Table 2 adds four ablation runs, one at $W = 1$ and three at $W = 2$ with $m = 1$, 16 and 32. Every run starts from the same weights and trains for 8 epochs on the same corpus with the same optimizer. Runs differ only in the objective, W, ρ and m.

Objective. Our runs train with the objective of Section 3.2, with $m = 4W$ draws per window, so 8, 12, 16 and 20 at $W = 2$ to 5. Each draw is a latent of dimension $d_z = 128$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, drawn afresh at every round. The scheduled-sampling rate ρ is 0.25 or 0.5 and stays constant through training. Training windows start at every response position, so $s = 1$. Each target carries the weight $1/\kappa_t$ of Equation 4. Equation 5 then divides by the number of response tokens, so the loss is a mean over response tokens. The loss is computed over 96 windows at a time, which changes only the order of summation.

Optimizer. Every run fine-tunes the same Qwen3-8B snapshot in bf16, with the input embeddings, the output head and the final layer norm frozen. BAdam (Luo et al., 2024) trains one transformer layer at a time and moves to the next layer every 22 steps, in ascending order. Its learning rate is a constant 10^{-5}, with no weight decay. Our runs first train only the projection matrix $\mathbf{W}_z$ for 792 steps, with the backbone frozen (Section 3.2). Appendix A.1 gives the group size and the update steps of a training round.

Compute. Training examples are at most 2,816 tokens long, within the training context, so none is truncated. Every W run trains on 4 L40S GPUs, and the AR fine-tuned model on one L40S. One step of a W run takes 10.66, 13.52, 17.20 and 19.77 s at $W = 2$, 3, 4 and 5, and one step of the AR fine-tuned model takes 2.3 s.

C Evaluation Details

This appendix gives the protocol behind the accuracy cells of Section 4.

C.1 Benchmarks and Metrics

Protocol. All six generated benchmarks follow the protocol of Nemotron-LD in NeMo-Skills (Fu et al., 2026). That protocol is zero-shot, with the question as a chat user turn, greedy decoding, non-thinking mode and at most 4,096 new tokens, on every problem of these sets. Every row of Table 1 decodes one problem per call (batch 1), for accuracy and speed alike. Answers that reach the 4,096-token cap are kept and graded as written. The original row of Table 1 is our run of the released Qwen3-8B under this protocol.

Grading. One grader scores every answer of a benchmark, whichever system wrote it. The NeMo-Skills math grader scores GSM8K and MATH-500 on the last boxed answer. The EvalPlus base tests (Liu et al., 2023b) score HumanEval and MBPP, the latter as MBPP+ v0.2.0. LiveCodeBench v6 here is the 454 problems from August 2024 through April 2025, graded by its own test runner, and IFEval by its official checker at prompt-level strict accuracy. MMLU (Hendrycks et al., 2021a) (5-shot) and HellaSwag (Zellers et al., 2019) (0-shot, on its validation set) are scored by likelihood with lm-eval 0.4.12 (Gao et al., 2024).

C.2 Seeds and Repeats

Seeds. Each generated-benchmark cell of a W row is the mean of three seeds. One-token cells and the MMLU and HellaSwag cells are read once. Under greedy decoding a seed sets only the latent stream, since nothing else in the decoding loop is random (Section 3.3). Each problem's latents come from a stream seeded by the run's seed and the problem's id, so batch size and problem order change no draw.

Seed spread. A W row draws latents, so we check how much its cells move from one seed to another. Across the three seeds, each GSM8K, MATH-500 and HumanEval cell of $W = 2$ to 5 moves little, with a population standard deviation of at most 0.29 points. The three-seed mean of each of these cells is therefore a stable reading.

D Speed Measurement Protocol

This appendix describes how every speed cell of Section 4 is measured.

D.1 Prompt Set

The six timed tasks are the six generated benchmarks. Every row, ours and each baseline's, is timed on every problem of each set in Appendix C.1. Every system reads the same prompts, rendered as in Appendix C.1, with the same 4,096-token cap. Each speed-up divides by the system's own base model, timed on those same sets. For every row built on Qwen3-8B, that base is the original Qwen3-8B in our one-token loop (Appendix E.2).

D.2 Timing Procedure

Timing. Every timed row of Table 1 runs alone on one NVIDIA L40S, at batch 1, with greedy decoding in bf16. Each prompt is timed after untimed warm-up runs, with the same numbers of warm-up runs and timed repeats for every row. A row's tokens per second is its generated tokens

Table 3: Decoding at another width W' (GSM8K accuracy in %, higher is better). Each row decodes one trained model at width W', with nothing retrained, on the full GSM8K test set. The diagonal $W' = W$ repeats Table 1. At $W' = 1$ a model decodes one slot per pass with a latent, while the one-token decoding of Table 2 (D) uses no slot and no latent.

	$W' = 1$	$W' = 2$	$W' = 3$	$W' = 4$	$W' = 5$
$W = 2$ model, decode width W'	91.41	91.23	9.02	2.83	0.00
$W = 3$ model, decode width W'	91.21	91.10	90.93	6.82	1.54
$W = 4$ model, decode width W'	90.19	90.14	90.09	89.89	12.59
$W = 5$ model, decode width W'	86.68	86.61	86.58	86.56	86.35

summed over a task's prompts, divided by its seconds summed over the same prompts, with the prefill inside the timed region. A second clock times decoding alone, without the prefill (Appendix E.2).

Reported measures. A speed-up is the ratio of two rows' tokens per second on the same prompts. It is never a ratio of seconds per answer, which would credit a row for writing shorter answers. Mean $\times$ is the arithmetic mean of a row's six per-task speed-ups. Tokens per forward pass (TPF) counts committed tokens over forward passes, with the prefill counted as one pass. Leaving the prefill out would change our TPF values only slightly. Pass cost is the time of a window pass divided by the time of a one-token pass.

E Additional Results

This appendix extends Sections 4.2 and 4.3 with further accuracy and speed results and an analysis of errors.

E.1 Accuracy

Tables 3, 4 and 5 decode the trained models in other ways, with nothing retrained.

Decoding at another width. Table 3 decodes each trained model at widths it was not trained for, to see whether another width helps. Decoding wider than the trained width fails, since the slots beyond W were never trained. Decoding narrower gains little accuracy and commits fewer tokens per pass. We therefore decode each model at the width it was trained for (Section 3.3).

Use of the latent. Section 3.1 makes the latent the only part of the model that couples the W tokens of a window, so we test whether the trained model uses it (Table 4). On a ground-truth prefix, two latent draws give different tokens in about a fifth of the windows at $W = 2$ and about a third at $W = 4$ (Table 4a). The same draw read twice changes no window, so every change comes from the latent. Across 32 draws of one window, the probabilities its slots give their ground-truth tokens tend to rise and fall together, so a draw shifts the slots of a window jointly. At decoding, one draw held for the whole answer costs little accuracy, while a latent set to zero lowers it sharply (Table 4b). A zero vector lies far from the training draws, whose norm is near $\sqrt{d_z} = \sqrt{128} \approx 11.3$, so the slots then receive an input they never saw in training. The model therefore uses its latent to couple the tokens of each window, as Section 3.1 intends. It also needs the latent drawn from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, as in training.

Sampling. Qwen3 recommends sampling in its non-thinking mode (Yang et al., 2025), so Table 5 checks whether window decoding also works with sampling, at its temperature 0.7 and top-p 0.8. Each slot samples its token from its own distribution given the window's latent, so the method needs no change. In every row, the AR fine-tuned model included, sampled accuracy stays close to greedy accuracy. Window decoding therefore works with sampling as it does with greedy decoding.

Table 4: Use of the latent by the $W = 2$ and $W = 4$ models, with nothing retrained.

Reading	$W = 2$	$W = 4$
windows that change, %	19.6	34.2
slots move together across draws, Spearman's r_s	0.52	0.49
control, the same draw twice, windows that change, %	0	0

(a) Whether the latent changes what the model writes. Two latent draws decode the same ground-truth prefix over the full GSM8K test set, with the most likely token at each slot. A window changes when at least one of its slots commits another token, and the control reads the same draw twice. Spearman's r_s is the rank correlation between the log-probabilities that the slots give their ground-truth tokens across 32 draws of one window, averaged over windows. Slots responding to the latent independently would give an r_s near 0.

W	Latent at decoding	Math GSM8K	Math MATH-500	Code HumanEval	Mean
$W = 2$	a fresh draw per window (the recipe)	91.23	82.07	78.66	83.99
	one draw held for the whole answer	90.95	81.87	78.25	83.69
	the latent set to zero	80.19	69.53	63.82	71.18
$W = 4$	a fresh draw per window (the recipe)	89.89	80.87	76.63	82.46
	one draw held for the whole answer	89.61	80.67	76.22	82.17
	the latent set to zero	67.50	57.20	51.22	58.64

(b) How the latent is drawn at decoding (accuracy in %, higher is better). The same models are decoded three ways, and Mean is over GSM8K, MATH-500 and HumanEval, as in Table 2.

Table 5: Decoding with sampling (accuracy in %, higher is better), on the full GSM8K and HumanEval test sets. Sampling uses temperature 0.7 and top-p 0.8, Qwen3's non-thinking values without its top-k, and each sampled cell averages three seeds. The greedy columns repeat Table 1.

Run	GSM8K greedy	GSM8K sampled	HumanEval greedy	HumanEval sampled
AR fine-tuned model, one token	92.27	92.44	80.49	80.08
$W = 2$, window	91.23	91.05	78.66	78.25
$W = 3$, window	90.93	91.03	78.05	77.64
$W = 4$, window	89.89	89.61	76.63	76.02
$W = 5$, window	86.35	86.00	72.15	71.54

E.2 SPEED

Speed-up per benchmark. Figure 7 shows the speed-up of IMLE-LLM on each generated benchmark, beside the baselines built on Qwen3-8B (values from Table 1).

Speed of our one-token loop. Table 1 divides the speed of each row built on Qwen3-8B by that of the original Qwen3-8B decoded one token at a time in our own loop, so we check whether this loop is a slow reference. With every answer forced to the same length, the original Qwen3-8B runs at 43.0 tokens per second in our loop and 41.06 in Hugging Face Transformers (Wolf et al., 2020). On the same forced answers, vLLM 0.22.0 (Kwon et al., 2023), SGLang (Zheng et al., 2024) and LMDeploy 0.10.1 (LMDeploy Contributors, 2023) with its PyTorch engine run it at 46.2, 46.5 and 44.6 tokens per second. Our loop is a little faster than Hugging Face Transformers and a little slower than the three serving engines. It is therefore a fair reference for Table 1, not a slow one.

Answer length. A model that writes shorter answers finishes sooner, so we compare the length of window answers with that of one-token answers of the same weights. Across the six timed tasks and $W = 2$ to 5, window answers are about as long as one-token answers, at 99 to 100 % of their length. The speed-up therefore does not come from shorter answers.

Where the speed-up comes from. A $W = 5$ window writes up to five tokens per pass, so it could at best decode $5\times$ faster than one token at a time. Each $W = 5$ pass commits 4.70 tokens on average (Table 1). The pass cost of a $W = 5$ window on one L40S at batch 1 is 1.14, on average over the

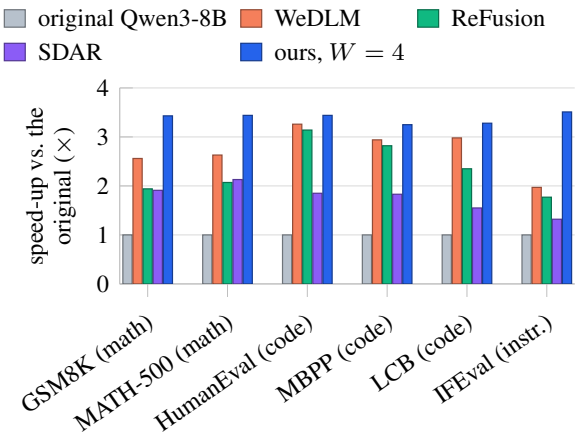


Figure 7: Speed-up over the original Qwen3-8B on each generated benchmark at $W = 4$, for IMLE-LLM (ours) and the baselines built on Qwen3-8B (higher is better).

Table 6: Speed-up at batch 1, 4 and 16 (higher is better). Each cell is the speed-up of a W model over the same weights decoded one token at a time at the same batch, averaged over the six timed tasks on one L40S. At batch 1 the values are Table 1's Mean ×. Batch 4 and batch 16 run over all prompts of the sets in Appendix C.1.

Run	batch 1	batch 4	batch 16
$W = 2$, window	1.73×	1.69×	1.58×
$W = 3$, window	2.55×	2.49×	2.26×
$W = 4$, window	3.39×	3.28×	2.86×
$W = 5$, window	4.09×	3.97×	3.29×

six timed tasks. We therefore expect a speed-up of $4.70/1.14 = 4.12\times$, and we measure $4.09\times$ (Table 1). The speed-up is thus set by the tokens each pass commits and by the small extra cost of a window pass.

Measurement checks. We also check whether the speed-up comes from how it is measured. Timed on decoding alone, without the prefill, the speed-up is a little higher than in Table 1, so the prefill does not create it. Window answers reach the 4,096-token cap about as often as the original model's answers, so capped answers do not lift the speed-up. A paired bootstrap over the timed prompts puts the 95 % interval of every W row's speed-up within ± 0.5 % of its value. The speed-ups of Table 1 are therefore not an artifact of the prefill, the token cap or prompt-to-prompt variation.

Batch size. Table 6 asks how the speed-up over the same weights decoded one token at a time holds when several prompts are decoded together. At batch 1 this speed-up is the same as Table 1's Mean ×, since each of our W models, decoded one token at a time, runs as fast as the original Qwen3-8B. The speed-up falls as the batch grows, and more so at larger W. Decoding a single prompt leaves most of the GPU's compute idle. A window pass, which handles $2W$ positions per prompt, fills that spare compute at a smaller batch than a one-token pass. Window decoding still keeps most of its speed-up at batch 16.

Size and memory. A faster decoder should not need a larger model or much more memory, so we check both. IMLE-LLM adds about 0.5 million parameters to Qwen3-8B, those of $\mathbf{W}_z$. We measure peak GPU memory on one L40S, with one forced 4,096-token answer per request, at the narrowest and widest windows, $W = 2$ and $W = 5$. It stays close to that of the original Qwen3-8B, at 16.0 against 15.9 GiB at batch 1 and 26.5 to 26.6 against 25.5 GiB at batch 16. Window decoding therefore adds little to the model's size and memory.

E.3 Error Analysis

Wrong answers. We sort the wrong GSM8K answers of each W model into classes, to see whether decoding in windows makes the answers repeat or stop early. We test the classes in the order repetitive, cut at the cap, empty, no extractable answer, and well formed but wrong. Each wrong answer takes the first class it meets. An answer is repetitive when a word repeats 4 times in a row, a character 20 times or a 6-word phrase 4 times. At every W, most wrong answers are well formed, 78 to 93 % of them. None of the wrong GSM8K answers is empty or cut at the cap. The share of repetitive answers stays at the AR fine-tuned model's level. Decoding in windows therefore adds no empty or capped answers and no extra repetition.

Accuracy along the window. Later slots predict further ahead in the same pass, so we check how often each slot of the $W = 4$ model predicts its ground-truth token. Teacher-forced on the ground-truth GSM8K solutions, with one latent draw per window, slot 1 does so 74.3 % of the time. Slot 4, which predicts three tokens further ahead in the same pass, does so 52.1 % of the time. Yet the $W = 4$ model's GSM8K accuracy barely changes between decoding one slot per pass and decoding four (Table 3). The lower token accuracy of the later slots therefore costs little answer accuracy.