PROVE: PROPOSER-VERIFIER DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding accelerates autoregressive (AR) language models by letting a small model draft tokens serially and a large model verify them in parallel. We reverse these roles. Starting from a standard AR checkpoint, we fine-tune the large model into a proposer that predicts a block of future tokens in a single forward pass, then use one additional forward pass to verify the proposed block. This gives ProVe, proposer-verifier decoding. The verifier can be a smaller AR model or the proposer itself; the latter gives ProVe-lossless, which reproduces the proposer's greedy AR decoding. Block proposals already provide substantial room for acceleration: with a block size of 16, the proposed block has an average AR-matching prefix of 6.6 tokens on GSM8K. Across six benchmarks with a 32B proposer, ProVe-lossless reaches $2.5\times$ average AR throughput without training a separate draft head. With a small 3B verifier, ProVe reaches $2.7\times$ average throughput at 2.6 accuracy points below AR on the six-task average, providing a controllable tradeoff between speed and accuracy. [1]

1 INTRODUCTION

Autoregressive (AR) language models produce one token per forward pass: a response takes as many serial forward passes as it has tokens. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023) cuts the number of passes of the large model by adding a small model. A small draft model proposes a sequence of tokens, one at a time. The large target model scores the sequence in one forward pass and keeps the proposed tokens up to the first one it disagrees with. The gain is set by how well the small model predicts the large one. Yet the draft model has to be much smaller than the target for its serial passes to be cheap. The roles are thus fixed by cost: drafting is serial, so it goes to the cheap model; verifying is one pass, so it goes to the expensive one.

We ask whether the usual assignment of roles is still necessary once the large model can propose multiple tokens in a single forward pass. MARS (Jin et al., 2026) provides exactly this capability: starting from an ordinary AR checkpoint, it fine-tunes the model into a *block proposer* that predicts the next B tokens in one causal forward pass. The first proposed token is exactly the model's ordinary next-token prediction, while later positions are predicted without access to the intervening future tokens (Section 2). Proposal quality decreases with depth in practice (Figure 2), but the key systems consequence is that proposal is no longer a chain of B serial model calls.

This changes the role assignment itself. Once the large model can produce a candidate block in one pass, the small model no longer has to spend serial steps drafting it; instead, it can read the realized block once and judge how much to keep. This gives **ProVe**, **Pro**poser-**Ve**rifier decoding (Figure 1): the large block-proposing model proposes, and a verifier reads the proposed block in a second pass before deciding how much of it to commit. In the default ProVe setting, a smaller AR model serves as the verifier. In ProVe-lossless, the proposer verifies its own block, so the resulting sequence reproduces the proposer's greedy AR decoding. Both variants replace many serial AR steps with exact two forward passes per decoding cycle.

Why pay for a verifier when proposer confidence is already available for free? MARS commits from the same proposed block using the proposer's own confidence, requiring no additional model pass. This gives a controlled comparison: the proposer, the candidate block, and its confidence are held fixed, and ProVe changes only what happens after the block has been proposed. We find that the free confidence rule leaves much of the usable prefix uncommitted: along greedy continuations of

[1] Our training code, serving engine implementation and checkpoints are available at `URL`.

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

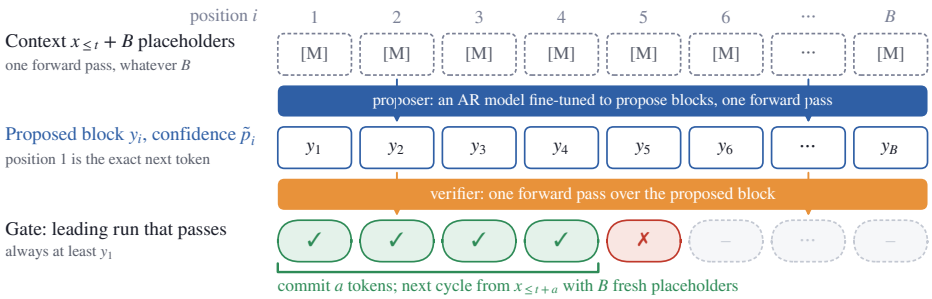

Figure 1: One ProVe cycle. The proposer predicts a block of candidate tokens from placeholders in one forward pass. A second pass verifies the proposed block before the next cycle begins.

GSM8K problems, the proposed block has an average AR-matching prefix of 6.6 tokens, while the default confidence rule commits only 3.0. Lowering the confidence threshold commits more tokens, but also increases the rate of committing tokens that diverge from greedy AR decoding (Section 3.1). The central question is therefore whether one additional verification pass can recover enough of this unused prefix to repay its cost.

We evaluate ProVe and ProVe-lossless with a 32B instruction proposer over six benchmarks and with a 32B reasoning proposer (Section 4). ProVe-lossless reproduces the proposer's greedy AR decoding and reaches $2.5\times$ AR throughput on average, ranging from $1.4\times$ on short constrained text to $4.0\times$ on templated chains of thought. It outperforms a public EAGLE-3 head and nearly matches a head fine-tuned specifically for the proposer, without training a separate draft head. With a smaller verifier, ProVe provides a controllable speed–accuracy tradeoff: a 7B verifier reaches $2.3\times$ average throughput with mean accuracy within 0.3 points of AR, while a 3B verifier reaches $2.7\times$ at 2.6 points below AR.

Contributions.

- Building on the block-proposing capability of MARS, we reverse the usual speculative-decoding roles: the large model proposes B tokens in one pass, and a verifier reads the realized block in one more. The verifier can be a smaller AR model or, in ProVe-lossless, the proposer itself (Section 3).
- We isolate the value of verification with a controlled comparison to the original MARS decoding rule: the proposer and proposed block are identical, while the confidence-only rule is free and ProVe deliberately pays one additional pass to read the realized trajectory. On GSM8K, the block has an average AR-matching prefix of 6.6 tokens, while the default confidence rule commits only 3.0 (Section 3.1).
- At 32B scale, ProVe-lossless reproduces the proposer's greedy AR decoding at $2.5\times$ average throughput without a separate draft head, while smaller verifiers provide a tunable speed–accuracy tradeoff reaching $2.7\times$ average throughput with a 3B verifier (Section 4.1).

2 Preliminaries: Block-Proposing AR Models

ProVe builds on a block-proposing AR model: an ordinary decoder-only AR model that is additionally trained to predict the next B tokens in one forward pass. We obtain this proposer following MARS (Jin et al., 2026). Appending B placeholders to the context allows one causal forward pass to produce B predictions for the next B tokens. When no placeholders are present, the model is decoded as an ordinary AR model.

Block proposal. Let P_{P} denote the proposer and let `[MASK]` denote its placeholder token. Given a context $x_{\le t}$ followed by B copies of `[MASK]`, a single causal forward pass produces B distributions over the next B tokens:

$$\tilde{p}_i(v) = P_{\mathrm{P}}\left(v \,\middle|\, x_{\le t}, \underbrace{\texttt{[MASK]}, \ldots, \texttt{[MASK]}}_{i-1}\right), \qquad i = 1, \ldots, B. \tag{1}$$

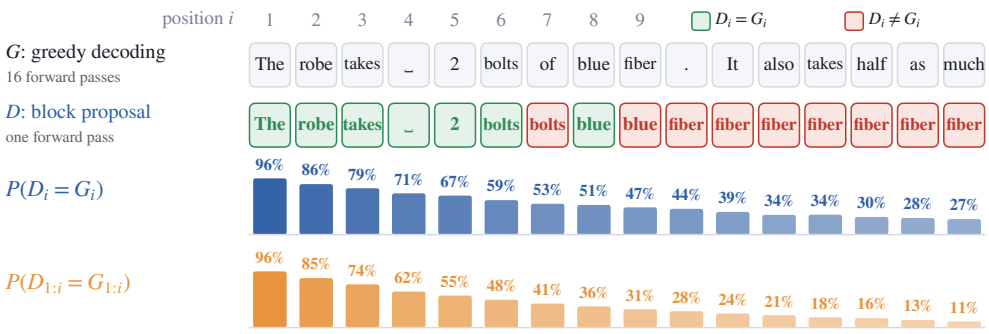

Figure 2: Block proposal quality decays with depth. Top: an example with $B = 16$, comparing the proposer's one-pass block proposal D with its token-by-token greedy continuation G. Bottom: over 437 proposed blocks from 64 GSM8K continuations, both the probability that an individual position matches, $P(D_i = G_i)$, and the probability that the entire prefix through that position matches, $P(D_{1:i} = G_{1:i})$, decrease with position.

The first distribution, $\tilde{p}_1$, is exactly the proposer's ordinary AR next-token distribution. The asymmetry across positions is important. For $i = 1$, no future token is missing from the conditioning context. For $i = 2$, the proposer predicts one step further without seeing the token at position $t + 1$; for larger i, progressively more intervening future tokens are replaced by placeholders. Deeper positions therefore solve an increasingly difficult prediction problem within the same forward pass. The proposal is obtained by greedy readout, $y_i = \arg\max_v \tilde{p}_i(v)$, and we use $\tilde{p}_i \equiv \tilde{p}_i(y_i)$ to denote the confidence assigned to the proposed token. As Figure 2 shows, proposals are most reliable near the beginning of the block and become less reliable with depth in practice.

Training. Following MARS (Jin et al., 2026), training combines the ordinary next-token objective with a block-prediction objective on placeholder-corrupted responses. The former preserves ordinary AR generation, while the latter teaches the model to predict multiple future tokens in one pass. Importantly, block proposal does not introduce a separate draft network or prediction head: ordinary AR prediction and block prediction use the same transformer and output head. The placeholders only provide the input pattern on which the model learns to predict farther into the future. Full training details are given in Appendix A.

Original decoding rule. MARS uses the proposer's own confidence to decide how much of each block to commit. It always commits the first proposed token, then continues from left to right while the confidence remains above a threshold τ. The committed prefix is appended to the context, and the proposer generates a fresh block from the new position. Thus, each cycle uses one forward pass and commits between one and B tokens. Section 3.1 examines how much of the proposed block this confidence rule is able to use.

3 Proposer-Verifier Decoding

We take the MARS block proposer as fixed and change what happens after it produces a block. The original MARS rule commits from the proposer's confidence, which is already available from the proposal pass. ProVe deliberately spends one additional forward pass so that a verifier can read the realized proposed tokens themselves before deciding how much to commit. This makes MARS a particularly clean control: the proposer and candidate block are unchanged, and the only new computation is verification. We first ask whether this extra pass provides information worth paying for, then describe two choices of verifier.

3.1 Why Verify the Proposed Block?

A block proposal is useful until it first diverges from the proposer's greedy AR continuation. To measure how much useful content the proposer already produces, we run it along its own greedy

continuations on randomly selected 300 GSM8K problems and propose a block of 16 tokens at every position. We compare each proposed block with the continuation obtained by token-by-token greedy decoding. The average AR-matching prefix is 6.6 tokens.

The original MARS rule is a strong baseline precisely because its confidence signal is free: $\tilde{p}_i$ is produced by the same forward pass that produces the candidate y_i. With its default confidence threshold (τ=0.95), however, it commits only 3.0 tokens per block on average. Lowering the threshold commits more tokens, but also increases the frequency of committing a token that diverges from greedy AR decoding. The proposed block therefore contains useful tokens that cannot be recovered simply by relaxing the free confidence rule. For an additional verifier pass to be worthwhile, it must identify enough of this unused matching prefix to compensate for its own latency.

The verifier provides an additional signal. Given the context and a proposed block $y_1, \ldots, y_B$, an AR verifier V assigns each proposed token the probability

$$q_i = P_V(y_i \mid x_{\leq t}, y_1, \ldots, y_{i-1}), \qquad i = 1, \ldots, B. \tag{2}$$

All B scores are obtained in one teacher-forced forward pass. Unlike the proposer's confidence, which is computed from placeholders, the verifier scores each token after reading the actual prefix.

The two scores, $\tilde{p}_i$ and q_i, answer different questions. The proposer confidence $\tilde{p}_i$ asks whether y_i is plausible at that future position before the intervening future tokens have been instantiated. The verifier probability q_i asks whether the same token remains plausible after the concrete proposed prefix $y_{<i}$ has been revealed. Conceptually, $\tilde{p}_i$ plays a prior-like role by exposing the proposer's own preference, while q_i is trajectory-conditioned on the proposal that actually materialized. A token can therefore look plausible to the proposer in advance yet become inconsistent with the particular trajectory proposed before it.

This also explains why the verifier can be much smaller than the proposer. The proposer must be capable of predicting several future positions while the intervening tokens remain unknown, whereas the verifier receives those tokens through teacher forcing. The verifier is not asked to reproduce the proposer's long-horizon prediction ability; it only judges whether a proposed continuation remains locally plausible after the missing context has been filled in.

The proposer confidence is still useful because it records how strongly the proposer itself preferred the candidate. We therefore treat $\tilde{p}_i \cdot q_i$ as a simple agreement score: it is high only when the candidate is both strongly supported by the proposer and compatible with the realized proposed prefix according to the verifier. Figure 3 compares the two signals and their product on the same proposed blocks. Verifier probability separates AR-matching from non-matching proposals better than proposer confidence, with AUC 0.960 versus 0.910. Their product reaches an AUC of 0.964 and performs comparably to the best hand-designed combinations we tested. Because the product is simple and parameter-free, we use it as the gate score in ProVe.

3.2 Verification

ProVe decodes in repeated propose–verify cycles. Given the current context, the proposer first generates a block of B candidate tokens in one forward pass. The verifier then reads the context together with the proposed block in one teacher-forced pass. A prefix of the proposed block is committed, and the next cycle starts from the extended context. There are two ways to verify the proposed block.

ProVe: a smaller model verifies. Let $\tilde{p}_i$ be the proposer's confidence in its i-th proposed token and q_i the verifier probability from Equation 2. We score each proposed token by their product, $\tilde{p}_i \cdot q_i$. For a threshold θ, we commit the longest leading prefix whose tokens pass the threshold:

$$a = \max\{t \in \{1, \ldots, B\} : \tilde{p}_i \cdot q_i \geq \theta \text{ for all } 2 \leq i \leq t\}. \tag{3}$$

The first token is always committed because it is the proposer's ordinary next-token prediction. A higher threshold is more conservative; lowering it commits longer prefixes but can also commit tokens that diverge from greedy AR decoding. The verifier never generates tokens itself. It only decides where to cut the proposed block.

ProVe-lossless: the proposer verifies itself. A special case is to use the proposer as its own verifier. After proposing a block, we feed the proposed tokens back to the proposer as ordinary

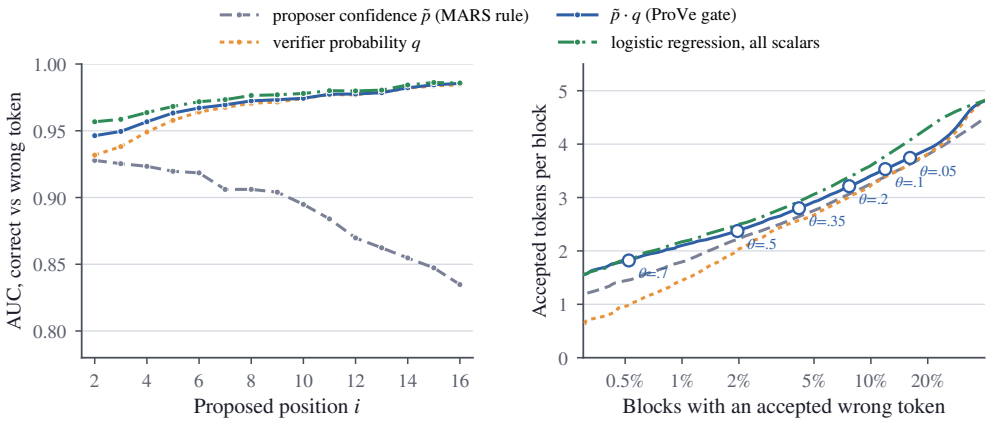

Figure 3: Left: AUC for distinguishing proposed tokens that match the greedy AR token, by proposed position with a 3B verifier on GSM8K. Right: committed tokens per block against the fraction of blocks in which the gate commits a non-matching token as each score's threshold is swept.

tokens in a second forward pass. This pass gives the proposer's greedy next-token prediction after every proposed prefix.

We compare the proposal with these predictions from left to right. As long as they agree, the proposed prefix is exactly the prefix that ordinary greedy AR decoding would have produced. At the first disagreement, we commit the proposer's greedy prediction instead and end the cycle. If all B proposed tokens agree, we additionally commit the next greedy token produced after the full block.

Every token committed by the cycle therefore lies on the proposer's greedy AR trajectory. This gives a simple induction argument for exactness: if the context at the start of a cycle is a greedy-AR prefix, matching proposed tokens keep the decoder on that same prefix, and the first disagreement is replaced by exactly the greedy next token. If the whole block matches, the token predicted after the full block is again the next greedy token. Repeating the argument across cycles reproduces the proposer's greedy AR decoding token for token. ProVe-lossless therefore requires *no threshold and no second model*; each cycle consists of exactly two proposer forward passes.

3.3 Cost

The advantage of reversing the two roles comes from where the serial work is paid. Let c_P and c_V denote the cost of one proposer and verifier pass, respectively, and define $\rho = c_V/c_P$. Normalizing one proposer pass to unit cost, the per-cycle costs C are

$$C_{ProVe} = 1 + \rho, \qquad C_{ProVe-lossless} = 2, \qquad C_{\text{Spec}} = 1 + k\rho, \tag{4}$$

where speculative decoding uses the same small model for k serial drafting steps. If a cycle commits a tokens, its idealized speedup over ordinary AR decoding is a/C.

The key difference is the factor multiplying ρ. Speculative decoding invokes the small draft model k times per cycle, whereas ProVe invokes the verifier only once. Increasing the proposal block size can expose more candidate tokens without adding another serial model call. A larger verifier can therefore be affordable in regimes where using the same model for k serial draft steps is not. For example, the 3B model in our implementation has $\rho \approx 0.34$. The idealized per-cycle cost is then about 1.34 proposer-pass equivalents for ProVe, but $1 + 5\rho \approx 2.70$ when the same model is used for five serial speculative-drafting steps. The small model has not become cheaper; its cost is simply paid once per proposed block rather than once per serial draft step. Actual speedup also depends on how many tokens each method commits per cycle.

Equation 4 ignores the small additional cost of processing a block and serving overheads. Measured pass costs and implementation details are given in Appendix A.

4 EXPERIMENTS

There are three decoding rules making different use of the same block proposer. MARS uses one proposer pass and decides how far to proceed from the proposer's placeholder-based confidence alone. ProVe adds one pass of a smaller model that reads the realized proposed block, yielding a tunable speed–accuracy tradeoff. ProVe-lossless instead spends a second proposer pass to recover the exact greedy AR trajectory. The experiments below evaluate whether the extra verification pass pays for itself in throughput, and how the choice of verifier changes the resulting operating point.

Setup. Our main proposer is Qwen2.5-32B-Instruct (Qwen et al., 2025), adapted into a block-proposing AR model with $B = 16$ following MARS (Jin et al., 2026). To isolate the incremental effect of block training, we use a two-stage controlled pipeline: one epoch of ordinary AR SFT on the same instruction data, followed by one epoch of block-proposal training. The stage-1 AR checkpoint is a control rather than a requirement of ProVe; in practice, block training can continue from an already post-trained checkpoint using matched in-domain or post-training data. We use Qwen2.5-3B-Instruct and 7B-Instruct as verifiers, either as stock checkpoints or fine-tuned on generations from the proposer. Appendix A reports training details and the effect of block training on ordinary AR quality.

Baselines. We compare against five decoding baselines: standard autoregressive (AR) decoding, ProVe-lossless, the original MARS confidence rule, EAGLE-3 (Li et al., 2025), and standard speculative decoding. AR performs ordinary greedy decoding with the same fine-tuned 32B model used as the proposer in our method, without block prediction or verification. We use it as the reference point for both task accuracy and decoding throughput. For ProVe, we evaluate confidence thresholds $\theta \in \{0.10, 0.05, 0.01\}$, where θ is applied to the combined confidence score $\tilde{p}_i \cdot q_i$. The original MARS confidence rule serves as a no-extra-pass control: it uses the same proposer and produces the same candidate blocks as ProVe, but directly commits the proposed tokens without an additional verification pass. For EAGLE-3, we used both its publicly released draft head and a draft head fine-tuned specifically for our 32B proposer. Finally, we evaluate standard speculative decoding using the same stock 3B or 7B model as a serial draft model, with the 32B model serving as the verifier. This provides a controlled comparison in which the small model is held fixed while its role is reversed relative to our setup. We use speculative draft lengths $k \in \{3, 5\}$ and report sensitivity to k in Appendix A. All methods use greedy decoding.

Evaluation. We evaluate on the full test splits of HumanEval, GSM8K, GPQA, BBH, MMLU-Pro, and IFEval using the lm-evaluation-harness (Gao et al., 2024), with up to 1024 generated tokens. These benchmarks span short constrained outputs and substantially longer structured continuations, which is useful because block-decoding speed depends on how long a proposed continuation remains predictable. Accuracy and throughput are measured from the same generations. All methods run in the same SGLang-based serving stack (Zheng et al., 2024), with a single request in flight and the 32B proposer tensor-parallelized over two GPUs. Each configuration is compared with its own same-node AR run, and we report throughput relative to that reference. Additional serving details and evaluation uncertainty are given in Appendices A and D.

4.1 MAIN RESULTS

ProVe-lossless provides strong exact acceleration. Table 1 shows that ProVe-lossless reaches $2.54\times$ AR throughput on average while reproducing the proposer's greedy AR decoding. The speedup ranges from $1.43\times$ on IFEval to $3.98\times$ on BBH. The wide range already suggests that acceleration is controlled by the structure of the generated continuation rather than by a fixed model-side speedup; Figure 5 returns to this variation below. ProVe-lossless requires neither a second model nor a separately trained draft head. For comparison, the public EAGLE-3 tree reaches $2.32\times$ on average, while an EAGLE-3 head fine-tuned specifically for our proposer reaches $2.58\times$. Thus, self-verification provides acceleration comparable to an external draft head tahat explicitly trained on proposer's output.

ProVe trades accuracy for additional speed. Using a smaller verifier introduces a controllable speed–accuracy tradeoff through the threshold θ (Figure 4). With the stock 7B verifier at $\theta =$

Table 1: Six general benchmarks, 32B instruction proposer. Exact decoders report relative throughput only, because their algorithmic score change is zero. Lossy cells report score changes relative to greedy AR (percentage points) / throughputs relative to AR. Spec uses the stock 3B or 7B model as its draft ($k = 5$); ProVe uses the same model as its verifier. Absolute task scores are reported in Appendix B, and evaluation uncertainty in Appendix D.

Method	Small model	HumanEval		GSM8K		GPQA		BBH		MMLU-Pro		IFEval		Avg	
AR (reference)	–	82.3	68 tok/s	89.5	70 tok/s	36.2	69 tok/s	84.0	69 tok/s	60.1	69 tok/s	83.2	69 tok/s	72.6	1.00×
Exact (output equals AR; relative throughput only)															
ProVe-lossless	–		2.42×		2.86×		2.16×		**3.98×**		2.37×		1.43×		2.54×
EAGLE-3, chain of 5	head		2.20×		1.84×		1.77×		2.26×		1.98×		1.37×		1.90×
EAGLE-3, tree of 60	head		2.86×		2.33×		2.17×		2.59×		2.36×		1.59×		2.32×
EAGLE-3 fine-tuned, chain of 5	head		2.43×		2.29×		2.06×		2.45×		2.08×		1.45×		2.13×
EAGLE-3 fine-tuned, tree of 60	head		**3.08×**		**3.00×**		**2.38×**		2.83×		**2.52×**		**1.68×**		**2.58×**
Spec k=5	3B stock		1.85×		1.47×		1.64×		2.07×		1.60×		1.16×		1.63×
Spec k=5	7B stock		1.54×		1.22×		1.42×		1.77×		1.33×		1.08×		1.39×
Lossy (Δ score | relative throughput)															
MARS τ=0.95	–	0.0	1.27×	0.0	1.49×	−2.0	1.28×	−1.7	2.43×	−0.6	1.38×	−1.1	1.03×	−0.9	1.48×
MARS τ=0.90	–	−1.2	1.38×	−0.6	1.66×	−2.3	1.36×	−2.6	2.70×	−0.7	1.50×	−1.7	1.07×	−1.5	1.61×
ProVe θ=0.10	3B stock	−0.6	2.47×	+0.2	2.33×	−0.4	1.98×	−2.2	3.62×	−1.3	2.11×	−2.2	1.17×	−1.1	2.28×
ProVe θ=0.05	3B stock	−1.8	2.73×	−1.3	2.65×	−0.2	2.17×	−2.3	3.78×	−1.5	2.25×	−2.8	1.35×	−1.6	2.49×
ProVe θ=0.01	3B stock	−1.8	**2.85×**	−3.0	2.89×	−1.1	**2.52×**	−3.6	3.95×	−3.3	2.55×	−2.6	**1.58×**	−2.6	2.72×
ProVe θ=0.10	3B trained	+0.6	2.57×	−1.4	2.88×	−4.0	2.15×	−1.9	3.79×	−2.0	2.25×	−2.6	1.29×	−1.9	2.49×
ProVe θ=0.05	3B trained	0.0	2.70×	−1.4	3.00×	−2.2	2.22×	−2.5	3.86×	−2.4	2.39×	−2.2	1.37×	−1.8	2.59×
ProVe θ=0.01	3B trained	−1.8	2.76×	−3.8	**3.11×**	−3.6	2.49×	−3.8	**3.99×**	−4.1	**2.59×**	−2.4	1.54×	−3.2	**2.75×**
ProVe θ=0.10	7B stock	+2.4	2.24×	+1.0	2.33×	−4.0	1.87×	−1.1	3.52×	−0.5	1.99×	−1.1	1.19×	−0.6	2.19×
ProVe θ=0.05	7B stock	+2.4	2.35×	+1.1	2.42×	−0.9	2.03×	−1.3	3.60×	−0.8	2.12×	−1.8	1.29×	−0.2	2.30×
ProVe θ=0.01	7B stock	+0.6	2.46×	+0.2	2.59×	−4.5	2.32×	−1.9	3.72×	−1.6	2.33×	−1.8	1.48×	−1.5	2.48×
ProVe θ=0.10	7B trained	+1.8	2.48×	−1.2	2.76×	−2.7	2.04×	−1.6	3.68×	−1.8	2.20×	−0.9	1.26×	−1.1	2.40×
ProVe θ=0.05	7B trained	+1.2	2.53×	−1.1	2.84×	−2.0	2.14×	−2.3	3.76×	−2.4	2.30×	−2.4	1.35×	−1.5	2.49×
ProVe θ=0.01	7B trained	+0.6	2.62×	−3.3	3.00×	−2.2	2.39×	−3.5	3.85×	−3.7	2.50×	−2.6	1.48×	−2.4	2.64×

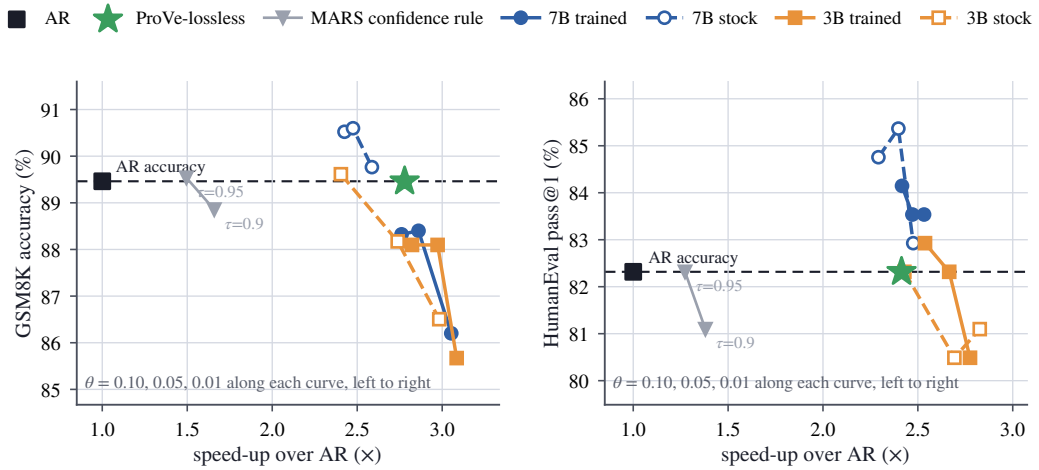

Figure 4: Speed against accuracy on the full GSM8K test set (left, $n = 1319$) and HumanEval (right, $n = 164$). The square is AR; the star is ProVe-lossless, placed on the AR accuracy it reproduces; the grey triangles are the confidence rule of MARS at $\tau \in \{0.95, 0.90\}$; each colored curve is ProVe with one verifier at $\theta \in \{0.10, 0.05, 0.01\}$, dashed for the stock model and solid for the verifier trained on the proposer's outputs. All speedups use same-node AR references; evaluation uncertainty is reported in Appendix D.

0.05, ProVe reaches 2.30× average throughput at only −0.3 points relative to AR. Lowering the threshold commits longer prefixes and increases speed: with a stock 3B verifier at $\theta = 0.01$, ProVe reaches 2.72× average throughput at −2.6 points. On GSM8K, the same configuration reaches 2.89× throughput at −3.0 points relative to AR. ProVe-lossless provides an exact reference point at 2.54× average throughput.

Fine-tuning the verifier improves acceptance efficiency. Fine-tuning the verifier on generations from the proposer changes not only its ranking of proposed tokens but also the scale of its probabilities. A fixed value of θ can therefore correspond to different levels of conservativeness before

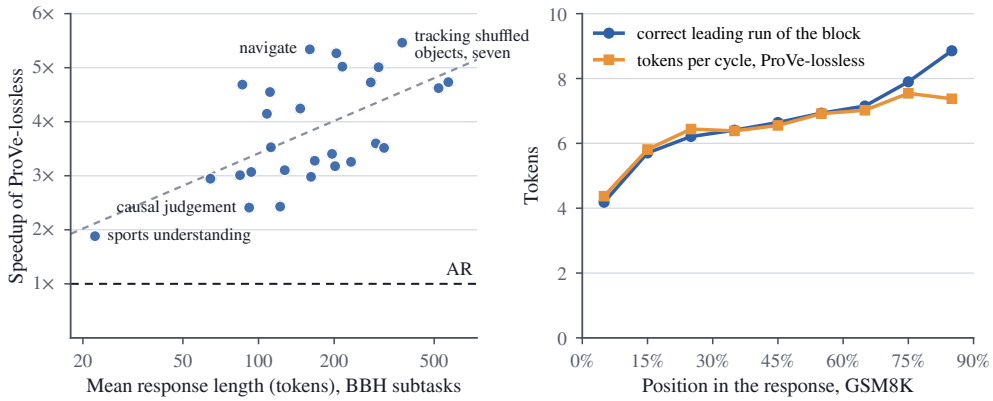

Figure 5: Where the gain comes from. Left: ProVe-lossless speedup on each of the 27 BBH subtasks against the subtask's mean response length (log scale; the same-node runs of Table 1), with the fit in log length dashed. Right: a block of 16 proposed at every position of 300 GSM8K greedy continuations, showing the AR-matching prefix length of the proposed block and the tokens ProVe-lossless commits per cycle when replayed along the trace, by relative position in the response.

and after fine-tuning. To separate ranking quality from this calibration shift, we compare stock and trained verifiers at matched rates of committing a non-matching token. At matched error rates, the fine-tuned verifiers commit 6–10% more tokens per cycle than their stock counterparts. We provide the full comparison in Appendix C.

ProVe avoids serial drafting cost. At the conservative $\theta = 0.10$ operating point, ProVe is faster than speculative decoding with the same small model. With the stock 3B model, speculative decoding reaches $1.63\times$ average throughput, while ProVe reaches $2.28\times$ at -1.2 points relative to AR. With the stock 7B model, the corresponding throughputs are $1.39\times$ and $2.19\times$, with ProVe at -0.6 points. Sweeping the speculative draft length over $k \in \{3, 5\}$ does not change this comparison; Appendix A reports the full sensitivity. The difference is largest on longer structured generations such as BBH and GSM8K. This matches the cost analysis in Section 3.3: speculative decoding calls the small model repeatedly to draft tokens, whereas ProVe calls it once to verify one proposed block.

Longer responses are usually faster. Figure 5 shows that the speedup tracks how much of a proposed block can be committed. Across the 27 BBH subtasks, ProVe-lossless speedup increases from about $1.9\times$ on the shortest responses to $5.5\times$ on the longest, with a correlation of $r = 0.59$ with log response length. The same mechanism appears within individual GSM8K responses: the average AR-matching prefix grows from 4.2 tokens near the beginning of a response to 8.9 tokens near the end, while ProVe-lossless commits 4.4 to 7.5 tokens per cycle. The method therefore benefits most when the continuation becomes predictable enough for a single proposal to match greedy AR for several positions.

4.2 A REASONING PROPOSER

We next test whether the same behavior extends to long chain-of-thought generation. This setting is particularly informative because long reasoning traces provide more opportunities for multi-token acceleration, while decision points can also make future tokens less predictable. We use Qwen3-32B in thinking mode as the proposer and the stock Qwen3-0.6B as the verifier; the same 0.6B model serves as the draft model for speculative decoding. See Appendix A for proposer training details.

On GSM8K, ProVe-lossless reaches $2.47\times$ AR throughput, while ProVe at $\theta = 0.10$ reaches $2.34\times$ at -0.4 points relative to AR. On MATH-500, ProVe-lossless reaches $2.79\times$, and ProVe at $\theta = 0.10$ reaches $3.16\times$ at -0.2 points. Speculative decoding with the same 0.6B model reaches $2.06\times$ and $2.28\times$ on the two benchmarks, respectively. At $\theta = 0.05$, ProVe roughly matches ProVe-lossless in throughput on GSM8K at -1.2 points and reaches $3.39\times$ on MATH-500 at -3.8 points.

Table 2: Reasoning proposer on GSM8K and MATH-500. Lossy cells report score change relative to greedy AR (percentage points) / throughput relative to AR; exact decoders show relative throughput only. Absolute scores are reported in Appendix B, and evaluation uncertainty in Appendix D.

Configuration	GSM8K (1319)		MATH-500 (500)	
AR (reference)	94.9	73.2 tok/s	91.6	70.4 tok/s
Exact (relative throughput only)				
Spec, $k = 5$	2.06×		2.28×	
ProVe-lossless	**2.47**×		**2.79**×	
Lossy (Δ score | relative throughput)				
ProVe, $\theta = 0.10$	−0.4	2.34×	−0.2	3.16×
ProVe, $\theta = 0.05$	−1.2	**2.45**×	−3.8	**3.39**×

5 Related Work

Speculative decoding and draft heads. Speculative decoding drafts k tokens with a cheap model and verifies them with the target in one pass, preserving the target distribution (Leviathan et al., 2023; Chen et al., 2023); later work cuts the draft cost with heads trained on the target's hidden states and tree-structured candidates (Li et al., 2024a;b; 2025), the target's own early layers (Zhang et al., 2024; Elhoushi et al., 2024), Jacobi-style n-gram candidates (Fu et al., 2024) or of a suffix automaton (Hu et al., 2025). Nearly all of this work is exact, the draft only changes speed, and the roles are fixed: drafter are designed to be small and cheap, while the verifier are large and expensive. Judge Decoding (Bachmann et al., 2025) and Big Little Decoder (Kim et al., 2023) relax the verifier's rule, as ProVe does with its threshold. Neither lets the large model propose.

Block proposals. Several methods let a model propose several positions in one of its own passes: PaSS (Monea et al., 2023) appends learned look-ahead embeddings, Parallel Prompt Decoding (Chen et al., 2025) trained prompt tokens, Medusa (Cai et al., 2024) extra heads, multi-token prediction (Gloeckle et al., 2024; DeepSeek-AI et al., 2025) extra output modules, and Set Block Decoding (Gat et al., 2025) and MARS (Jin et al., 2026) fine-tune the model to predict masked future tokens, the proposer of this paper. ReDrafter (Cheng et al., 2024), Amphista (Li et al., 2024c), FastEagle (Huang et al., 2026) and DFlash (Chen et al., 2026) draft a block in one pass of a trained *small* draft network under exact verification by the target, the standard direction with a block draft; IDLM (Yu et al., 2026) fuses the verification of one proposed block and the proposal of the next into one pass of the same model. Block diffusion models (Arriola et al., 2025), DSpark (Cheng et al., 2026) and MARS commit on the model's own confidence; what ProVe adds is the verifying pass that reads the proposed block. And through experiments, we show that the verification step is essential to commit more tokens.

6 Conclusion

In this work, we show that block proposals contain substantially more usable content than the proposer's own confidence suggests: on GSM8K, the proposed block has an average AR-matching prefix of 6.6 tokens, while the original confidence rule commits only 3.0. The gap arises because proposer confidence is computed from placeholders and therefore does not see the realized proposed prefix. A verifier instead reads the proposed tokens themselves, providing a second signal for deciding how much of the block to commit. This simple change gives ProVe two useful operating alternatives. With a smaller verifier, ProVe provides a controllable speed–accuracy tradeoff; when the proposer verifies itself, ProVe-lossless reproduces greedy AR decoding with no second model and reaches 2.5× average AR throughput. Our results suggest that block proposal and verification need not follow the traditional speculative-decoding division of labor. A large model can propose several tokens in parallel, while verification can be performed in one additional pass.

REFERENCES

Marianne Arriola, Aaron 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 *The Thirteenth International Conference on Learning Representations*, 2025.

Gregor Bachmann, Sotiris Anagnostidis, Albert Pumarola, Markos Georgopoulos, Artsiom Sanakoyeu, Yuming Du, Edgar Schönfeld, Ali Thabet, and Jonas Kohler. Judge decoding: Faster speculative sampling requires going beyond model alignment. In *International Conference on Learning Representations*, 2025.

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 *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235, 2024.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Hao Mark Chen, Wayne Luk, Yiu Ka Fai Cedric, Rui Li, Konstantin Mishchenko, Stylianos Venieris, and Hongxiang Fan. Hardware-aware parallel prompt decoding for memory-efficient acceleration of LLM inference. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. In *Forty-third International Conference on Machine Learning*, 2026.

Xin Cheng, Xingkai Yu, Chenze Shao, Jiashi Li, Yunfan Xiong, Yi Qian, Jiaqi Zhu, Shirong Ma, Xiaokang Zhang, Jiasheng Ye, Qinyu Chen, Chengqi Deng, Jiping Yu, Damai Dai, Zhengyan Zhang, Yixuan Wei, Yixuan Tan, Wenkai Yang, Runxin Xu, Yu Wu, Zhean Xu, Xuanyu Wang, Muyang Chen, Rui Tian, Xiao Bi, Zhewen Hao, Shaoyuan Chen, Huanqi Cao, Wentao Zhang, Anyi Xu, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation, 2026. URL https://arxiv.org/abs/2607.05147.

Yunfei Cheng, Aonan Zhang, Xuanyu Zhang, Chong Wang, and Yi Wang. Recurrent drafter for fast speculative decoding in large language models. *arXiv preprint arXiv:2403.09919*, 2024.

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

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

Mostafa Elhoushi, Akshat Shrivastava, Diana Liskovich, Basil Hosmer, Bram Wasti, Liangzhen Lai, Anas Mahmoud, Bilge Acun, Saurabh Agarwal, Ahmed Roman, et al. Layerskip: Enabling early exit inference and self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12622–12642, 2024.

Yichao Fu, Peter Bailis, Ion Stoica, and Hao Zhang. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 14060–14079, 2024.

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

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. *arXiv preprint arXiv:2509.04185*, 2025.

Fabian Gloeckle, Badr Youbi Idrissi, Baptiste Roziere, David Lopez-Paz, and Gabriel Synnaeve. Better & faster large language models via multi-token prediction. In *Forty-first International Conference on Machine Learning*, 2024.

Yuxuan Hu, Ke Wang, Xiaokang Zhang, Fanjin Zhang, Cuiping Li, Hong Chen, and Jing Zhang. Sam decoding: Speculative decoding via suffix automaton. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12187–12204, 2025.

Haiduo Huang, Jiangcheng Song, Wenzhe Zhao, and Pengju Ren. Fasteagle: Cascaded drafting for accelerating speculative decoding. In *ICASSP 2026-2026 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4111–4115, 2026.

Ziqi Jin, Lei Wang, Ziwei Luo, and Aixin Sun. Mars: Enabling autoregressive models multi-token generation, 2026. URL https://arxiv.org/abs/2604.07023.

Sehoon Kim, Karttikeya Mangalam, Suhong Moon, Jitendra Malik, Michael W Mahoney, Amir Gholami, and Kurt Keutzer. Speculative decoding with big little decoder. *Advances in Neural Information Processing Systems*, 36:39236–39256, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pp. 19274–19286. PMLR, 2023.

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

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-2: Faster inference of language models with dynamic draft trees. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 7421–7432, 2024b.

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

Zeping Li, Xinlong Yang, Ziheng Gao, Ji Liu, Zhuang Liu, Dong Li, Jinzhang Peng, Lu Tian, and Emad Barsoum. Amphista: Accelerate llm inference with bi-directional multiple drafting heads in a non-autoregressive style. *arXiv preprint arXiv:2406.13170*, 2024c.

Giovanni Monea, Armand Joulin, and Edouard Grave. Pass: Parallel speculative sampling. *arXiv preprint arXiv:2311.13581*, 2023.

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

Yifan Yu, Yuqing Jian, Junxiong Wang, Zhongzhu Zhou, Donglin Zhuang, Xinyu Fang, Sri Yanamandra, Xiaoxia Wu, Qingyang Wu, Shuaiwen Leon Song, Tri Dao, Ben Athiwaratkun, James Zou, Fan Lai, and Chenfeng Xu. Introspective diffusion language models, 2026. URL https://arxiv.org/abs/2604.11035.

Jun Zhang, Jue Wang, Huan Li, Lidan Shou, Ke Chen, Gang Chen, and Sharad Mehrotra. Draft& verify: Lossless large language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody H Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.

A Implementation Details

Serving. All configurations run in one SGLang-based stack (Zheng et al., 2024) extended with the ProVe algorithms, on two data-center GPUs with tensor parallelism of two for the 32B model, bf16 weights, the FlashInfer attention backend and a single request in flight; the AR controls use the unmodified server. ProVe is a decoding algorithm with a small configuration: block size 16, the product rule of Equation 3 with its threshold, the verifier checkpoint and the fraction of device memory reserved for the in-process verifier (its weights, KV cache and CUDA graphs): 0.18 for the 3B and 0.32 for the 7B, with the proposer's static pool at 0.70 and 0.55 so that the two never compete (an undersized verifier pool does not fail but silently degrades toward one accepted token per cycle; Appendix D). Each cycle is one scheduler step per pass, and proposed tokens that are not committed are dropped from the proposer's KV cache before the next cycle, so the cache after a cycle equals what AR decoding of the committed context would have produced. Spec uses the stack's standalone-draft implementation with the same 3B or 7B checkpoint; we evaluate serial draft lengths $k \in \{3, 5\}$ and report the sweep below. EAGLE-3 uses the stack's implementation with the head as a chain of five draft steps and with its default tree of 60 draft tokens (Table 1).

Measured pass costs. On the same two GPUs (60 GSM8K problems): one AR decode step 14.8 ms; one pass of the 32B over a 16-token window 15.6 ms, both as the proposing and as the verifying pass of ProVe-lossless; one in-process pass of the 3B verifier 5.0 ms ($\rho = 0.34$) and of the 7B 5.8 ms ($\rho = 0.40$). The verifier's pass is dominated by a fixed per-call cost of about 2.7 ms that does not depend on its size (the 0.5B costs 3.0 ms, halving the 7B's weight bytes with fp8 saves 0.2 ms, and running it without tensor parallelism is slower, 6.9 ms), which is why a stronger verifier is nearly free in Equation 4. Per-cycle scheduling and cache-trimming overhead is 8.7 ms in ProVe-lossless and 4 ms in ProVe, the part the equation omits; with it, $7.27/(2 \times 1.05 + 0.59) = 2.70\times$ against the measured $2.70\times$ for ProVe-lossless, and $4.13/(1.05 + 0.34 + 0.27) = 2.49\times$ against $2.49\times$ for ProVe with the 3B verifier at $\theta = 0.10$.

Block size. The proposer was trained with 16 placeholders but proposes a longer block when asked. On the full GSM8K test set ProVe-lossless commits 5.9 / 7.5 / 8.1 / 8.6 tokens per cycle at $B = 8/16/24/32$ and decodes at $2.44/2.79/2.84/2.84\times$: the speedup saturates from $B = 24$ on because both passes pay for every row of the window, and every other experiment uses the $B = 16$ the proposer was trained with.

The block objective and training. With t_b the position before block b and $\tilde{p}_i^{(b)}$ Equation 1 evaluated with the context $x_{\leq t_b}$, the loss is

$$\mathcal{L} \;=\; -\sum_j \log P_{\mathrm{P}}(x_{j+1} \mid x_{\leq j}) \;-\; \sum_b \sum_{i=1}^{B} \log \tilde{p}_i^{(b)}(x_{t_b+i})\,, \tag{5}$$

both streams in one forward pass over the concatenation of the placeholder stream and the original, with an attention pattern in which a placeholder sees the original tokens before its block and the placeholders before it; attention is causal everywhere. Both proposers use the reference implementation of Jin et al. (2026) on 32 GPUs: the instruction proposer 150 GPU hours for the AR stage and 384 for the block stage at 512 tokens per stream; the reasoning proposer 1,194 GPU hours at 16k tokens, a single-benchmark check because reasoning traces are long and block training doubles every sequence. The trained verifiers are the stock 3B and 7B fine-tuned for two epochs on the proposer's generations (about 4 and 10 GPU hours).

Effect of block training on ordinary AR quality. The instruction proposer is trained in two stages so that the incremental effect of the block objective can be measured cleanly. Stage 1 performs one epoch of ordinary AR SFT; stage 2 applies the MARS block-proposal objective on the same instruction data. Table 3 evaluates all three checkpoints with ordinary greedy AR decoding. The controlled comparison is therefore the stage-1 AR checkpoint against the final proposer, not the original Qwen2.5 checkpoint against the final proposer. Across the six benchmarks, block training changes the unweighted average from 73.1 to 72.5, a difference of -0.6 points; every per-task change is within 1.8 points.

Table 3: Ordinary greedy-AR accuracy before and after block-proposal training. The stage-1 AR-SFT checkpoint is the controlled reference for isolating the effect of the subsequent MARS block-training stage; the original Qwen2.5-32B-Instruct checkpoint is shown for context. Avg is the unweighted mean of the six tasks.

Checkpoint	HumanEval	GSM8K	GPQA	BBH	MMLU-Pro	IFEval	Avg
Original Qwen2.5-32B-Instruct	89.6	83.4	39.5	40.8	69.3	79.1	67.0
AR SFT (stage 1)	84.1	90.2	36.4	84.3	61.2	82.3	73.1
Final block proposer	82.3	89.5	36.2	84.0	60.1	83.2	72.6

The larger differences between the original checkpoint and stage 1 primarily reflect the ordinary SFT stage and its change in generation behavior, rather than the block objective. For BBH, the original checkpoint frequently emits paragraph breaks while the evaluation configuration treats `\n\n` as a stopping string; using EOS alone raises its score from 40.8 to 62.5, which is still below the stage-1 score of 84.3. On MMLU-Pro, stage-1 SFT reduces the mean generated length from 372 to 164 tokens, showing that the SFT data substantially changes response style before block training begins. We use stage 1 only to obtain a controlled reference under matched data; it is not an algorithmic requirement of ProVe. In practice, block-proposal training can be applied after an existing post-training stage using matched in-domain or post-training data.

Speculative draft length. We also vary the number of serial draft steps in speculative decoding, using the same stock 3B and 7B checkpoints as in the main comparison. Table 4 shows that neither $k = 3$ nor $k = 5$ dominates across tasks. With the 3B draft, $k = 5$ has the higher six-task average ($1.63\times$ versus $1.58\times$), whereas the 7B draft is essentially tied ($1.38\times$ versus $1.39\times$). Short constrained generations such as IFEval favor the shorter draft, while BBH benefits from the longer draft. The main comparison reports $k = 5$; the qualitative gap to ProVe is unchanged under $k = 3$.

Table 4: Sensitivity of speculative decoding to serial draft length k. The stock 3B or 7B checkpoint is used as the draft model; entries are throughput relative to the same-node AR reference.

Draft model	k	HumanEval	GSM8K	GPQA	BBH	MMLU-Pro	IFEval	Avg
3B stock	3	1.69×	1.52×	1.56×	1.85×	1.55×	1.28×	1.58×
3B stock	5	1.78×	1.56×	1.60×	2.07×	1.60×	1.18×	1.63×
7B stock	3	1.48×	1.30×	1.39×	1.65×	1.37×	1.17×	1.39×
7B stock	5	1.49×	1.25×	1.37×	1.78×	1.34×	1.06×	1.38×

Harness. Accuracy is scored by the lm-evaluation-harness fork of Jin et al. (2026) with the chat template applied, through a backend that sends one request at a time and records generated tokens and wall-clock time per request; throughput is total tokens over total in-flight seconds of a unit. Every server passes a fixed-prompt speed probe before use, or is restarted; models are staged to node-local disk before the server starts.

B Absolute Evaluation Scores

The main tables emphasize direct comparison by reporting score changes relative to the corresponding greedy-AR reference. For completeness, Tables 5 and 6 report the underlying absolute task scores. Positive deltas in the main tables indicate direction only; evaluation uncertainty is reported in Appendix D.

Table 5: Absolute task scores underlying the lossy block of Table 1. Avg is the unweighted mean of the six tasks. Exact decoders are omitted because their algorithmic target is the AR reference.

Method	Small model	HumanEval	GSM8K	GPQA	BBH	MMLU-Pro	IFEval	Avg
AR (reference)	–	82.3	89.5	36.2	84.0	60.1	83.2	72.6
MARS $\tau = 0.95$	none	82.3	89.5	34.2	82.3	59.5	82.1	71.6
MARS $\tau = 0.90$	none	81.1	88.9	33.9	81.4	59.4	81.5	71.0
ProVe $\theta = 0.10$	3B stock	81.7	89.6	35.7	81.9	58.7	81.0	71.4
ProVe $\theta = 0.05$	3B stock	80.5	88.2	35.9	81.7	58.6	80.4	70.9
ProVe $\theta = 0.01$	3B stock	80.5	86.5	35.0	80.4	56.7	80.6	70.0
ProVe $\theta = 0.10$	3B trained	82.9	88.1	32.1	82.1	58.1	80.6	70.7
ProVe $\theta = 0.05$	3B trained	82.3	88.1	33.9	81.5	57.7	81.0	70.8
ProVe $\theta = 0.01$	3B trained	80.5	85.7	32.6	80.2	56.0	80.8	69.3
ProVe $\theta = 0.10$	7B stock	84.8	90.4	32.1	82.9	59.6	82.1	72.0
ProVe $\theta = 0.05$	7B stock	84.8	90.6	35.3	82.7	59.2	81.3	72.3
ProVe $\theta = 0.01$	7B stock	82.9	89.6	31.7	82.1	58.5	81.3	71.0
ProVe $\theta = 0.10$	7B trained	84.1	88.2	33.5	82.4	58.3	82.3	71.5
ProVe $\theta = 0.05$	7B trained	83.5	88.4	34.2	81.7	57.7	80.8	71.0
ProVe $\theta = 0.01$	7B trained	82.9	86.2	33.9	80.6	56.4	80.6	70.1

Table 6: Absolute scores underlying Table 2. The measured ProVe-lossless score can differ slightly from AR under bf16 execution even though the decoding rule is algorithmically exact; Appendix D analyzes this effect.

Configuration	GSM8K	MATH-500
AR, proposer (reference)	94.9	91.6
Spec, $k = 5$	94.3	92.2
ProVe-lossless	94.0	91.6
ProVe, $\theta = 0.10$	94.5	91.4
ProVe, $\theta = 0.05$	93.7	87.8

C GATING RULES COMPARED OFFLINE

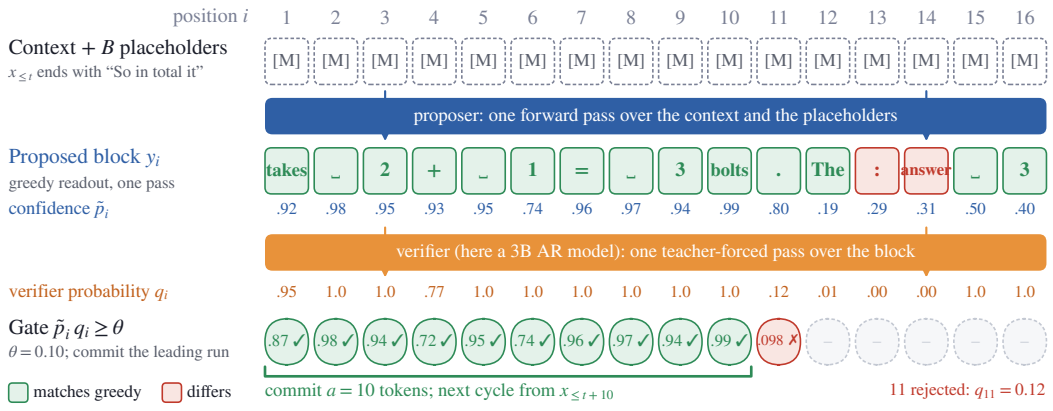

Figure 6: One ProVe cycle on a real block from a GSM8K answer (32B instruction proposer, 3B verifier). Green proposed tokens equal what the proposer generates one token at a time, red ones differ. The gate $\tilde{p}_i \cdot q_i \ge \theta$ with $\theta = 0.10$ commits ten tokens; the eleventh is correct but the verifier gives it 0.12, so the cycle stops there. The confidence rule of MARS at $\tau = 0.95$ would commit two, because $\tilde{p}_3 = 0.946$.

The study of Section 3.1 teacher-forces the proposer along its own greedy continuations of the first 300 GSM8K problems, proposes a block of 16 at every position (26,407 blocks), and records for every proposed token the proposer's top-two probabilities and entropy, the verifier's probability, top-two and entropy, and whether the token equals the greedy token. A sequential gate only judges tokens whose shallower siblings were correct, so statistics are compared on that population, by the AUC of ranking correct against wrong tokens and by the frontier a gate traces as its threshold is swept: accepted tokens per block against the fraction of blocks in which a wrong token was accepted (Table 7, Figure 3). The proposer's own confidence loses discrimination as the proposed block deepens, because it is computed under placeholder conditioning, and the product gains it, because the verifier sees the proposed tokens before the one it judges; the deep positions are where the accepted length is decided. Reweighting the factors, $\tilde{p}_i^{\alpha} q_i^{\beta}$, is temperature scaling in the log domain and moves the frontier by less than the noise between repeats; a logistic regression on all recorded scalars accepts 2 to 4% more tokens at matched divergence, the room left for a better scalar gate, which we do not deploy because the product has no parameters. Figure 3 in the main text plots the AUC by position and the swept frontier.

Table 7: Gating rules on the population a sequential gate sees (3B verifier, GSM8K, 26,407 blocks). AUC ranks correct against wrong proposed tokens; the remaining columns give accepted tokens per block, beyond the always-committed first token, at a matched fraction of blocks in which a wrong token was accepted.

Statistic	AUC	0.5%	1.0%	2.0%	5.0%
proposer confidence $\tilde{p}$ only (MARS)	0.909	2.44	2.77	3.21	3.73
verifier probability q only	0.960	1.93	2.42	3.02	3.63
$\tilde{p} \cdot q$ (Equation 3)	0.964	2.78	3.10	3.37	3.92
$\tilde{p}^2 \cdot q$	0.964	2.79	3.09	3.45	3.97
$(\tilde{p}_{(1)} - \tilde{p}_{(2)}) \cdot q$	0.961	2.77	3.06	3.40	3.97
$\min(\tilde{p}, q)$	0.957	2.71	2.93	3.28	3.74
logistic regression on all scalars (held-out prompts)	0.970	2.81	3.17	3.50	4.04

D MEASUREMENT VALIDITY

Exactness of ProVe-lossless. Feeding each generation of the main campaign back to the plain AR server with per-token log probabilities, the tokens of ProVe-lossless are the argmax at every position

except 0.84%, against 0.85% for the AR server's own generations scored the same way, with the same distribution of gaps to the argmax (median 0, maximum 0.4 nats): the two decoders differ only in bf16 tie-breaking. On the reasoning line the generations are long enough for tie-breaking to show: over 300 GSM8K problems, 20 generations are identical token for token, the first differing token sits at a median position of 176, and 266 of the 280 diverged pairs end at the same final answer.

Repeats and pooling. Repeats of the same configuration on different nodes agree in accuracy to the item and in throughput within 2%, except at the loosest thresholds of the 3B and 7B verifiers, which moved by 8 to 10% in opposite directions; every comparison in the paper is therefore decoded on one node in one job with its own AR run. Throughput is total generated tokens over total in-flight seconds of a unit; averaging per-request rates would overweight short generations, whose rate is dominated by prefill and the first cycle.

Evaluation Uncertainty We evaluate all methods on the full evaluation split of each benchmark. The test-set sizes and corresponding 95% confidence half-widths for the main six-benchmark evaluation are: HumanEval, $n = 164$ (± 5.9 points); GSM8K, $n = 1319$ (± 1.7); GPQA, $n = 448$ (± 4.4); BBH, $n = 6511$ (± 0.9); MMLU-Pro, $n = 12032$ (± 0.9); and IFEval, $n = 541$ (± 3.2). We therefore do not interpret accuracy differences within these ranges as meaningful differences between decoding methods. For the reasoning-model evaluation, MATH-500 contains 500 problems and has a 95% confidence half-width of approximately ± 2.4 points, which covers the accuracy differences among the configurations we report.

A silent failure. An undersized in-process verifier pool does not fail but degrades ProVe toward one accepted token per cycle; with the 7B verifier a fraction of 0.25 starved the pool (probe 119 instead of 172 tok/s) and the campaign uses 0.32 with the proposer's pool at 0.55.

Concurrency. Each ProVe cycle occupies more accelerator rows per accepted token than AR decoding, free only while the device is latency bound, so the advantage narrows with the number of concurrent requests C (Figure 7). On GSM8K prompts, throughput relative to AR at the same C falls from $3.2\times$ at $C = 1$ to $2.4\times$ at $C = 8$ and $1.6\times$ at $C = 32$ for ProVe-lossless, and from $3.3\times$ to $2.3\times$ and $1.3\times$ for ProVe (stock 3B verifier, $\theta = 0.01$), ahead of ProVe-lossless up to $C = 4$ and behind it from $C = 8$ on; the EAGLE-3 tree, at 60 rows per cycle, falls from $3.5\times$ to $1.9\times$ and reaches AR at $C = 32$.

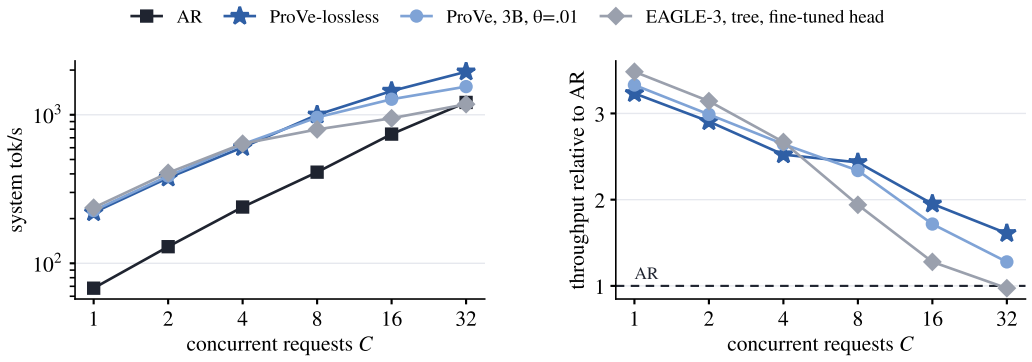

Figure 7: System throughput against the number of concurrent requests C on GSM8K prompts (up to 512 new tokens, about 140 generated per request), 32B instruction proposer, one two-GPU server per decoder, all four decoders on the same node. Left: generated tokens per second over all requests in flight; right: the same relative to AR at the same C.

E LIMITATIONS

Proposer training. The proposer requires block-specific fine-tuning and is therefore not a drop-in accelerator for an arbitrary checkpoint. Our proposer is also substantially under-trained relative

to the original MARS recipe because of our compute budget. MARS uses five epochs of ordinary AR training followed by five epochs of block training, whereas we use only one epoch of each, or 20% of the training in each stage. Despite this limited training, our proposer already produces an average AR-matching prefix of 6.6 tokens on GSM8K. More training data, longer block training, or incorporating the block-proposal objective during pretraining could substantially improve proposal quality and, consequently, the speedups of both ProVe variants.

Serving implementation. Our implementation is research-oriented and remains less optimized than industrial inference systems. The current serving stack incurs fixed overheads from scheduling, KV-cache management, and verifier calls that are not intrinsic to proposer-verifier decoding. These overheads reduce the realized speedup, particularly under concurrency. A production implementation with a more tightly integrated propose–verify loop, optimized cache management, and lower per-call overhead could translate longer committed prefixes more directly into end-to-end throughput.

Greedy decoding only. All reported experiments use greedy decoding. ProVe-lossless can in principle be extended to sampling using a speculative-sampling correction that preserves the target distribution, but we have not implemented or evaluated this setting in our serving system. The behavior and efficiency of ProVe under sampling therefore remain to be studied.

ProVe is not distribution-exact. Unlike ProVe-lossless, ProVe with a smaller verifier uses a thresholded acceptance rule and does not preserve the proposer's greedy AR output exactly. Its speed–accuracy tradeoff therefore depends on the verifier, threshold, and task, and we do not extrapolate the measured accuracy effects beyond the settings evaluated in this work.

Scope. Our experiments cover one model family (Qwen) and one proposer scale (32B). The instruction proposer is evaluated on six benchmarks, while the reasoning proposer is evaluated on GSM8K and MATH-500. Whether the same gains hold across substantially different model families and scales remains an open question.

F AI USE STATEMENT

In this work, we used generative AI tools to assist with the presentation and writing of proofs, propose or refine hypotheses, provide feedback on research methodology and experimental design, assist with method implementation, and interpret experimental results. We did not use generative AI tools to generate synthetic datasets, assist with translation, clean or reformat datasets, develop theoretical models or conceptual frameworks, formulate mathematical claims, or provide critical ingredients for proving mathematical claims. The use of generative AI for qualitative or thematic data analysis was not applicable to this work.

We also used generative AI tools to draft portions of the manuscript, improve the readability of the writing, and help identify relevant literature. All AI-assisted content was reviewed and verified by the authors. In particular, we carefully reviewed the manuscript and evaluated AI-assisted implementation using manually written unit tests. We take full responsibility for the final content of this work, including all text, claims, results, code, and other artifacts produced with the assistance of generative AI.