BitMTP: When Multi-Token Prediction Meets Low-Bit Large Language Models

Ning Zhang Shihao Wang Jinrui Zhang Chaodong Xiao Lei Zhang*
Department of Computing, The Hong Kong Polytechnic University
ning24.zhang@polyu.edu.hk, cslzhang@comp.polyu.edu.hk
{shihaow.wang,jin-rui.zhang,chaodong.xiao}@connect.polyu.hk

Abstract

For efficient deployment of large language models (LLMs), methods of ultra-low-bit backbones reduce weight storage, memory traffic, and linear-layer execution cost, while multi-token prediction (MTP) methods reduce sequential decoding steps by predicting multiple future tokens in parallel. Although complementary, the direct combination of these methods cannot accelerate decoding at comparable quality: masked drafting lacks intermediate token input supplied in next-token training, while wider computation and causal key–value (KV) updates can offset the savings from fewer steps. To address these challenges, we propose **BitMTP**, an algorithm–system co-design framework for efficient MTP on ultra-low-bit LLM backbones. A joint next-token prediction and MTP objective equips a single model with both autoregressive (AR) generation and block-causal drafting. Confidence-only decoding, speculative-only decoding, and hybrid decoding provide different quality–throughput operating points without using separate draft and verifier models. Accepted-prefix alignment keeps the committed prefix, materialized causal KV boundary, and CUDA Graph replay state consistent for continued generation. BitMTP uses this contract to select width-specialized low-bit execution and reuse captured graphs over shared KV storage. On NVIDIA H20, BitMTP with hybrid decoding achieves five-task mean speedups of $2.2\times$ on BitNet b1.58-2B and 2.1–$2.4\times$ on 1.7B–4B Bonsai models at comparable task quality. On RTX 3090 and Jetson Orin, it further achieves up to $2.0\times$ in five-task mean throughput and $3.3\times$ on MBPP+, respectively. These results demonstrate that BitMTP accelerates ultra-low-bit LLM generation across backbones, model sizes, and GPU types. The source code is released at `https://github.com/PolyU-VCLab/BitMTP`.

1 Introduction

The deployment of large language models (LLMs) is constrained by both model footprint and the serial decoding latency of autoregressive (AR) next-token prediction (NTP) [1–3]. To resolve this limit, ultra-low-bit models such as BitNet b1.58 reduce weight storage, memory traffic, and linear-layer execution cost through ternary weights [4–6], while multi-token prediction (MTP) methods target the sequential bottleneck by predicting multiple future tokens in one pass [7–13]. Despite their complementarity, direct combination of ultra-low-bit models and block-parallel MTP may not accelerate decoding at comparable quality: masked block prediction omits NTP's intermediate token inputs, calling for adaptation to preserve quality, while wider computation and acceptance-dependent causal key–value (KV) updates can offset the latency savings from fewer sequential decoding steps.

To better illustrate the above-mentioned problem, we apply the block-parallel prediction mechanism [10] to BitNet b1.58-2B [5] and name the fine-tuned model 'Naive BitNet+MTP'. On NVIDIA H20, we compare its five-task mean quality and decode throughput with the supervised fine-tuning

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

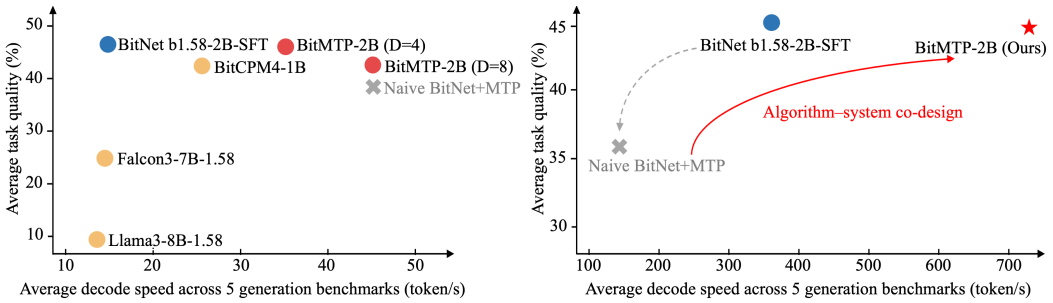


Figure 1: Five-task mean generation quality (%) and decode throughput (tokens/s) for BitNet b1.58-2B-SFT AR, Naive BitNet+MTP, and the proposed BitMTP on NVIDIA H20. The left panel shows the results by using BF16 reference execution, while the right panel uses low-bit W2A8 execution (2-bit weights and 8-bit activations). We see that the naive combination on BF16 improves throughput, but it loses both throughput and quality in the low-bit W2A8 setting. In contrast, our BitMTP substantially increases throughput while retaining task quality comparable to SFT AR.

(SFT) AR baseline of BitNet b1.58-2B. The left panel of Figure 1 shows that the naive combination improves throughput on BF16. However, we see in the right panel of Figure 1 that it loses both throughput and quality in the low-bit W2A8 setting (2-bit weights and 8-bit activations), which shows that fewer decoding steps alone cannot ensure faster low-bit generation. More generally, an MTP decoder's throughput depends on iteration cost and the number of committed tokens. First, a block decoder must execute a wider computation and maintain causal KV states for continued generation. Second, iteration cost is amortized over committed tokens, rather than all predicted tokens. The accepted prefix and decoding policy therefore determine the committed-token count, pending causal KV work, and subsequent replay input. Verifier-based policies additionally require a verification replay. We call the resulting changes in replay inputs, causal KV work, and verifier invocation 'accepted-prefix dynamics'. Efficiently executing these transitions turns parallel proposals into multi-token speedups on ultra-low-bit LLMs.

To achieve faster decoding at comparable quality on ultra-low-bit LLM backbones, we propose **BitMTP**, an algorithm–system co-design framework. At the training level, a joint NTP+MTP objective equips the model with mask-conditioned block-causal drafting and an AR path. At the decoding level, confidence-only decoding, speculative-only decoding, and hybrid decoding provide different quality–throughput operating points without using separate draft and verifier models. At the runtime level, three mechanisms are used to reduce block computation and state-update costs. M1 specializes packed low-bit execution by replay width and uses role-dependent output projections. M2 uses a finite-state CUDA Graph atlas to replay reachable pending-prefix states over shared KV storage. M3 prepares verifier inputs before the host branch decision in hybrid decoding. Accepted-prefix alignment keeps the committed prefix, materialized causal KV boundary, and CUDA Graph replay state consistent across decoding and runtime. This supports captured-graph reuse over shared KV history as accepted prefixes change, reducing overhead of multi-token decoding. As shown in Figure 1, BitMTP achieves substantially higher decode throughput at comparable task quality to the SFT AR baseline. Our main contributions are summarized below:

- **First end-to-end MTP framework for ultra-low-bit LLMs.** To our knowledge, BitMTP is the first framework to co-design MTP training, decoding, and runtime execution for ultra-low-bit LLMs. Accepted-prefix alignment coordinates decoding decisions with causal KV materialization and graph replay. Captured graphs reuse shared causal KV history across changing accepted prefixes, reducing full-history copying overhead and supporting faster low-bit multi-token decoding.
- **Joint training and flexible decoding.** BitMTP combines a joint NTP+MTP objective with confidence-only decoding, speculative-only decoding, and hybrid decoding. It thereby supports an AR path and mask-conditioned block-causal drafting without a separate draft–verifier pair. The three decoding policies provide different quality–throughput operating points.
- **Replayable runtime for dynamic MTP states.** BitMTP introduces a three-part runtime that reconciles accepted-prefix-dependent execution with fixed-shape CUDA Graph replay. Width-specialized low-bit execution, shared-KV graph replay, and hoisted verifier preparation control the kernel, memory, and scheduling overheads exposed by MTP on low-bit LLM backbones.

- **Validation across models and hardware types.** Experiments on BitNet b1.58-2B and Bonsai models from 1.7B to 4B demonstrate that BitMTP achieves mean decode speedups of up to $2.4\times$ on H20 and $2.0\times$ on RTX 3090. It also reaches a $3.3\times$ MBPP+ speedup on Jetson Orin, demonstrating generalization across backbones, scales, and GPU types.

2 Related Work

Low-bit LLMs. Low-bit LLMs can be obtained by quantizing a pretrained full-precision model or pretraining directly in the low-bit regime. Post-training quantization applies quantization after full-precision training [14, 15], whereas quantization-aware training incorporates the target quantizer during model optimization [6, 16]. For example, Bonsai applies quantization-aware training to pretrained Qwen3 backbones to obtain ternary models [17]. BitNet and BitNet b1.58 instead use native low-bit pretraining [4, 5, 18]. Although low-bit representations reduce weight storage and linear-layer execution cost, token-by-token AR generation still requires a separate model pass for each output token, limiting further throughput gains. Our BitMTP addresses this sequential bottleneck by predicting blocks and committing variable-length continuations on the low-bit backbone.

Parallel drafting and blockwise generation. Recent methods reduce sequential decoding steps through auxiliary heads, learned drafters, or block-parallel prediction [7, 8, 11, 19]. Medusa adds decoding heads and verifies alternative continuations with tree attention [8]. EAGLE predicts future hidden states, EAGLE-2 builds context-dependent draft trees, and EAGLE-3 predicts tokens from features fused across target layers [7, 20, 21]. Training approaches include future-summary supervision [22], non-adjacent targets [23], and self-distillation [24]. DeepSeek-V3 incorporates MTP during pretraining [13]. Masked-input adaptation with gated LoRA and consistency losses exposes a pretrained model's MTP potential [12]. Other methods reuse target-model components or adapt draft structures and budgets [9, 11, 25–27]. DFlash generates blocks with a target-feature-conditioned diffusion drafter before speculative verification [28]. Sequential Diffusion Language Models (SDLM) predict masked token blocks and use confidence to commit consecutive prefixes [10]. In addition, QSpec [29] shares 4-bit weights and a KV cache between W4A4 drafting and W4A16 verification. ML-SpecQD [30] pairs an MXFP4 draft with a BF16 target and recursively accelerates drafting with a smaller model. QuantSpec [31] combines hierarchical KV quantization with optional 4-bit draft MLP weights. Although these quantized draft–verify methods combine quantization with speculative decoding to reduce drafting and memory costs, sequential proposal dependencies and recurring verification still constrain further acceleration. Our BitMTP combines block-parallel proposals, conditional verification in hybrid decoding, and accepted-prefix-aligned low-bit execution to reduce work per committed token.

Inference runtimes for efficient generation. The CPU inference system of BitNet provides specialized kernels for ternary execution [32]. GPU systems such as Atom and QServe co-design low-bit quantization and serving optimizations [33, 34]. Other work optimizes attention and decode kernels [1, 35–37]. KV-cache techniques address allocation, compression, and quantization [38–42]. Serving systems improve request scheduling and execution [43–46]. For block generation on an ultra-low-bit LLM, the accepted prefix and verification branch determine causal KV materialization and the next replay input. Kernel, cache, and scheduling optimizations alone do not provide a coordinated end-to-end acceleration path for these transitions. Our proposed BitMTP provides this path through width-specialized packed operators, captured graphs over shared KV storage, and preparation of verifier inputs before the hybrid branch decision.

3 The BitMTP Framework

3.1 Method Overview

As illustrated in Figure 2, BitMTP enables low-bit multi-token generation through coordinated training, decoding, and runtime components. Joint training equips the model with both AR generation and mask-conditioned block-causal drafting. The decoding component uses confidence-only decoding, speculative-only decoding, or hybrid decoding to select the committed prefix. The runtime executes the selected transition with fixed-shape low-bit CUDA Graph replay. These components satisfy the same consistency requirement: the committed prefix, materialized causal KV boundary, and

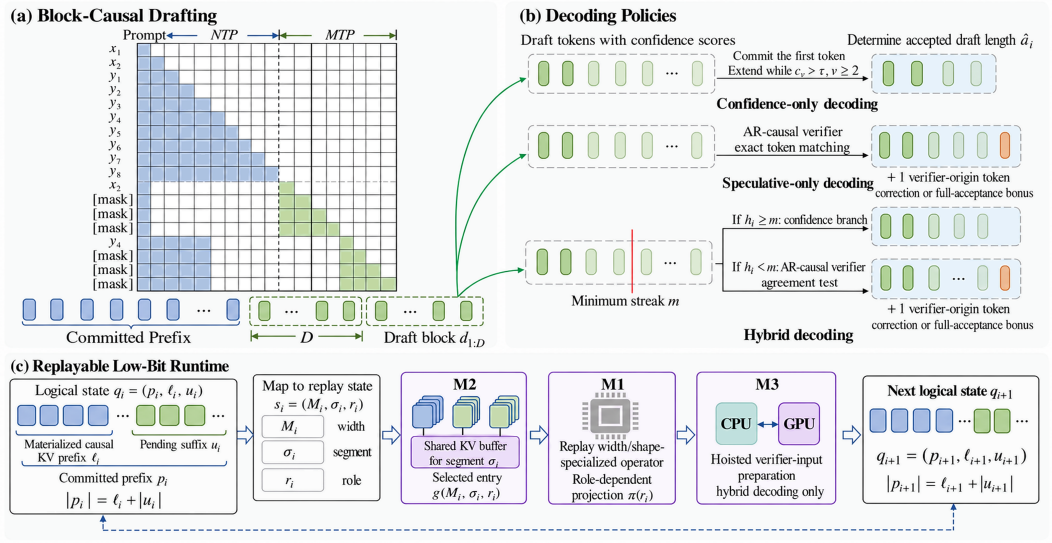


Figure 2: The BitMTP framework. (a) Joint NTP+MTP training supports mask-conditioned block-causal drafting and an AR path in one model. (b) Confidence-only decoding commits a confidence-selected prefix. Speculative-only decoding verifies each block, while hybrid decoding verifies when the confidence prefix is too short. Verifier-based steps add a correction or bonus token. (c) The runtime executes accepted-prefix transitions with width-specialized low-bit operators and captured CUDA Graphs. Graphs reuse causal KV history through shared storage. In hybrid decoding, verifier inputs are prepared before the host branch decision.

CUDA Graph replay state must remain mutually consistent among decoding iterations, which we call 'accepted-prefix alignment'.

Each draft block proposes D future tokens. At iteration i, decoding tracks $q_i = (p_i, \ell_i, u_i)$: the committed prefix p_i, its materialized causal KV length ℓ_i, and pending suffix u_i. The prefix includes the prompt and emitted tokens. Accepted-prefix alignment imposes the following length relation:

$$|p_i| = \ell_i + n_i, \tag{1}$$

where $n_i = |u_i|$ counts committed tokens awaiting causal KV. During initialization, prefill processes the prompt and builds its causal KV prefix. Since tokens emitted from the initialization logits have not yet been processed as causal inputs, they form the pending suffix. Later, u_i will contain confidence-committed drafts or a verifier-origin token, and the next draft replay computes their causal KV together with a new block.

The runtime maps q_i and the selected decoding policy to a replay state $s_i = (M_i, \sigma_i, r_i)$. The width M_i covers the pending suffix and the next block computation. The segment σ_i selects the active KV capacity, and r_i selects the numerical execution role. A verifier step additionally selects a width-$(D{+}1)$ replay with the verifier role. Each draft step computes and materializes the pending-token KV in q_i; the next state records committed tokens. Algorithm 1 in **Appendix A** summarizes one hybrid decoding iteration.

3.2 Block-Causal Drafting and Training

For a training sequence y anchored at s_b, BitMTP constructs the block input and targets:

$$X_b = [y_{s_b}, m_{\text{mask}}, \ldots, m_{\text{mask}}], \qquad Y_b = [y_{s_b+1}, \ldots, y_{s_b+D}], \tag{2}$$

where y includes the prompt and the response, and m_{mask} denotes the mask input. The observed anchor occupies the first input slot, and mask embeddings occupy the remaining $D{-}1$ slots. The targets in Y_b are the D ordered future tokens. Attention within the block is causal, as illustrated in Figure 2(a). Each prediction can attend to the anchor and the mask slots preceding its target position. Because the block inputs are constructed before the forward pass, the model produces the whole draft block in parallel.

We retain causal next-token supervision and combine it with block prediction. The summed losses are then normalized over all supervised targets:

$$\mathcal{L}_{\mathrm{mtp}} = \sum_{b\in\mathcal{B}} \sum_{u\in\mathcal{U}_b} -\log p_\theta(y_{s_b+u} \mid y_{<s_b}, X_{b,1:u}), \qquad \mathcal{L} = \frac{\mathcal{L}_{\mathrm{ar}} + \mathcal{L}_{\mathrm{mtp}}}{N_{\mathrm{ar}} + N_{\mathrm{mtp}}}, \tag{3}$$

where θ denotes the model parameters. The set $\mathcal{B}$ contains training blocks, and $\mathcal{U}_b \subseteq \{1, \ldots, D\}$ contains the valid supervised slots of block b. The slice $X_{b,1:u}$ comprises its first u input slots. The AR loss $\mathcal{L}_{\mathrm{ar}}$ sums causal next-token negative log-likelihoods, while N_{ar} and N_{mtp} count supervised AR and block targets. The AR term supports causal generation, while the MTP term learns block-causal proposals. The resulting model performs both drafting and AR verification.

3.3 Decoding Policies

BitMTP provides confidence-only decoding, speculative-only decoding, and hybrid decoding, as shown in Figure 2(b). Each policy operates on a draft block $d_{1:D}$ and determines the accepted draft length $\hat{a}_i$. A confidence step forms an output block of $\hat{a}_i$ draft tokens. A verifier step adds one verifier-origin token, giving $\hat{a}_i + 1$ tokens in the block.

Confidence-only decoding. With draft-token softmax probability c_u and threshold τ, the decoder always commits d_1 and extends the prefix through consecutive confident drafts:

$$h_i = 1 + \sum_{u=2}^{D} \prod_{v=2}^{u} \mathbf{1}[c_v > \tau]. \tag{4}$$

The h_i-token confidence streak is committed without verification, giving $\hat{a}_i = h_i$.

Speculative-only decoding. The same model processes $[\text{anchor}, d_1, \ldots, d_D]$ under an AR-causal mask, where the anchor is the last committed token. It produces verifier tokens $\tilde{d}_{1:D+1}$. Verification selects the longest matching prefix:

$$\hat{a}_i = \max\left\{k \in \{0, \ldots, D\} : d_u = \tilde{d}_u \text{ for every } u \le k\right\}. \tag{5}$$

The decoder forms an output block containing $d_{1:\hat{a}_i}$, followed by $b_i = \tilde{d}_{\hat{a}_i+1}$. The verifier-origin token b_i corrects the first rejected draft position or supplies a bonus token after full acceptance. Speculative-only decoding preserves the greedy continuation defined by the AR-causal verifier.

Hybrid decoding. Hybrid decoding uses a minimum confidence-prefix length m. It sets $\hat{a}_i = h_i$ and commits the confidence prefix when $h_i \ge m$. Otherwise, it invokes the AR-causal verifier to determine the accepted prefix and verifier-origin token.

3.4 Replayable Low-Bit Runtime

Figure 3 shows how M1–M3 implement a decoding transition. At each step, BitMTP selects a replay state $s_i = (M_i, \sigma_i, r_i)$ specifying the replay width, KV-capacity segment, and numerical role. M1 selects the packed operator and output projection, M2 replays the corresponding graph over shared causal history, and M3 moves verifier-input preparation ahead of the hybrid branch decision.

After a draft replay, the runtime copies the n_i pending-token KV rows into history. The selected decoding policy then updates the state as follows:

$$\begin{aligned} \bar{\ell}_i &= \ell_i + n_i, \\ q_{i+1} &= \begin{cases} \big(p_i \cdot d_{1:\hat{a}_i},\ \bar{\ell}_i,\ d_{1:\hat{a}_i}\big), & I_{\mathrm{verify}}(i) = 0, \\ \big(p_i \cdot d_{1:\hat{a}_i} \cdot b_i,\ \bar{\ell}_i + \hat{a}_i,\ [b_i]\big), & I_{\mathrm{verify}}(i) = 1, \end{cases} \end{aligned} \tag{6}$$

where $\bar{\ell}_i$ is the materialized KV length after processing the pending suffix, and b_i is the verifier-origin token from Section 3.3. The indicator $I_{\mathrm{verify}}(i)$ is 0 for confidence-only decoding, 1 for speculative-only decoding, and $\mathbf{1}[h_i < m]$ for hybrid decoding. Without verification, the committed draft prefix remains pending. With verification, its accepted KV rows are materialized and only b_i remains pending. Both branches preserve $|p_{i+1}| = \ell_{i+1} + |u_{i+1}|$.

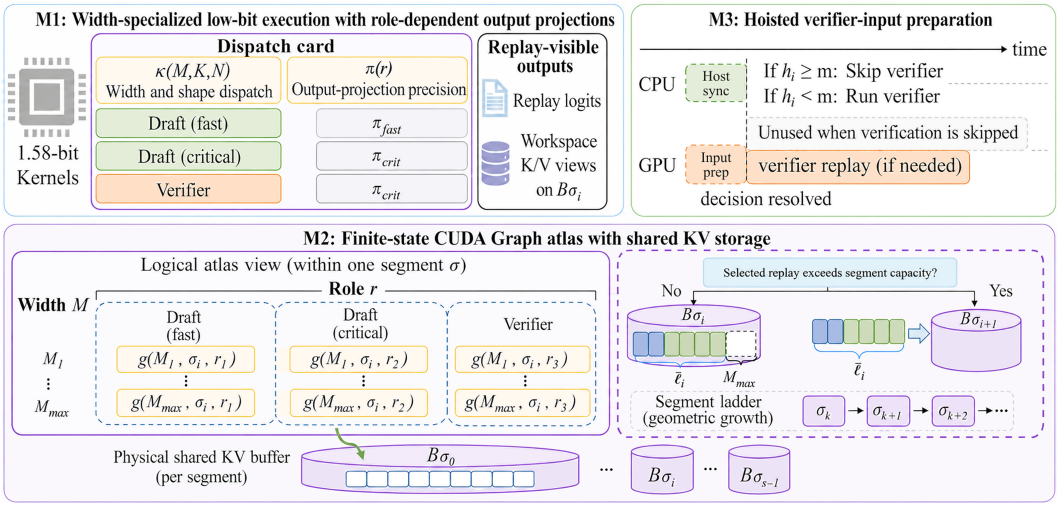


Figure 3: Runtime mechanisms M1–M3. M1 selects packed operators by replay width M and input/output dimensions (K, N), while numerical role r selects the output projection $\pi(r)$. M2 indexes captured graphs by (M, σ, r) and shares one KV buffer B_σ across widths and roles within each capacity segment. M3 moves verifier-input preparation ahead of the hybrid branch decision.

M1: Width-specialized low-bit execution with role-dependent output projections. M1 specializes packed low-bit execution for the replay widths used by AR generation, block drafting, and verification. As shown in the top-left panel of Figure 3, M1 selects a packed operator by (M, K, N) and an output projection by numerical role r. The dispatch is:

$$\kappa : (M, K, N) \longmapsto \mathcal{K}_{M,K,N}, \qquad \pi(r) = \begin{cases} \pi_{\text{fast}}, & r \in \mathcal{R}_{\text{fast}}, \\ \pi_{\text{crit}}, & r \in \mathcal{R}_{\text{crit}} \cup \mathcal{R}_{\text{ver}} \end{cases}, \tag{7}$$

where $\mathcal{K}_{M,K,N}$ is the selected packed operator, and K, N are its input and output feature dimensions. The set $\mathcal{R}_{\text{fast}}$ contains fast drafting roles, $\mathcal{R}_{\text{crit}}$ contains verification-critical drafting and AR roles, and $\mathcal{R}_{\text{ver}}$ contains verifier roles. Drafting selects its projection through r, while AR generation and verification use π_{crit}. All roles share the packed low-bit transformer body and project only the token rows required by the replay.

M2: Finite-state CUDA Graph atlas with shared KV storage. Accepted-prefix dynamics change both replay width and the materialized KV boundary, whereas CUDA Graph replay requires fixed shapes and addresses. As shown in Figure 3, M2 supports these changes with shared segment storage and captures a graph set $\mathcal{G}$ covering reachable transitions from a candidate graph set:

$$\mathcal{G}_{\text{cand}} = \{ g_{M,\sigma,r} : M \in \mathcal{M},\ \sigma \in \Sigma,\ r \in \mathcal{R} \}, \qquad \mathcal{G} \subseteq \mathcal{G}_{\text{cand}}, \tag{8}$$

where $g_{M,\sigma,r}$ denotes a graph with replay key (M, σ, r). The sets $\mathcal{M}$, Σ, and $\mathcal{R}$ contain supported replay widths, ordered KV-capacity segments, and M1 execution roles, respectively. The captured set covers reachable replay states, including width-$(D{+}1)$ verification.

Within segment σ, all graphs use fixed-shape views of one physical KV buffer B_σ, with $|\sigma|$ history slots and $M_{\max} = \max \mathcal{M}$ workspace slots. Width and role switches therefore preserve materialized history. After drafting, the runtime copies the pending-token KV rows into history; after verification, it promotes accepted draft rows, excluding the repeated anchor. These operations copy consecutive token rows within the segment. When a transition requires a larger capacity segment, the runtime copies the materialized history into the new buffer. Graphs within each segment thus reuse one causal KV history while implementing Eq. 6.

Shared KV storage is proportional to $\sum_{\sigma \in \Sigma} (|\sigma| + M_{\max})$ for a fixed model. With fixed replay widths, geometrically increasing capacities cover history length $\ell_{\max}$ using $O(\log \ell_{\max})$ segments and $O(\ell_{\max})$ KV slots. **Appendix A** details the replay masks and KV copies.

M3: Hoisted verifier-input preparation. M3 moves verifier-input preparation ahead of the branch decision in hybrid decoding, as shown in Figure 3. The draft block, committed anchor, and materialized causal KV boundary determine the verifier inputs before the host reads h_i. After drafting, the

runtime enqueues preparation of the width-$(D{+}1)$ verifier token buffer, positions, and AR-causal mask on the GPU stream, then obtains h_i on the host. Preparation is issued in each hybrid iteration, while verifier replay occurs only when $h_i < m$. The selected verifier graph therefore consumes the prepared inputs without a separate input-preparation stage after the branch decision.

4 Experimental Evaluation

4.1 Experimental Setup

Models, training, and decoding. We evaluate three ultra-low-bit LLM backbones: BitNet b1.58-2B [5], Bonsai-1.7B, and Bonsai-4B [17]. For each backbone, additional supervised fine-tuning produces the SFT model used as the AR baseline. The SFT and MTP variants use the same processed corpus, with MTP using the joint objective in Eq. 3. Generation uses greedy decoding. Primary confidence-only decoding and hybrid decoding use $\tau{=}0.9$. Hybrid decoding uses the model's AR-causal verifier to determine the accepted prefix when verification is required. **Appendix B** summarizes the training corpus and reports one-epoch training cost on Bonsai-1.7B.

Hardware and low-bit backends. Server, consumer, and edge tests use NVIDIA H20, RTX 3090, and Jetson Orin NX, respectively. The PyTorch/CUDA runtime uses packed W2A8 for decode-stage transformer linear layers, with INT2 weights for BitNet and group-128 ternary weights for Bonsai. H20 enables FP8 draft projections for the fast drafting roles in M1. The FP8 GEMM backend does not support the Ampere GPUs in RTX 3090 and Orin, so these platforms use non-FP8 draft projections. Orin runs under `MAXN_SUPER` with locked clocks. AR generation and verifier replay use verification-critical projections. Other baseline systems use their released runtimes.

Generation benchmarks. The generation suite covers IFEval for instruction following, GSM8K and MATH-500 for reasoning, and HumanEval+ and MBPP+ for code generation. We report strict instruction-following accuracy, exact match for reasoning, and EvalPlus pass@1 for code.

Metrics and model capability evaluation. Decode throughput divides timed output tokens by decode-loop time measured after prefill and request initialization. Task means are unweighted across the five generation benchmarks, and speedups use the matched SFT AR route on the same device and backend. The eleven-task capability evaluation uses `lm-eval` with full benchmark sets and standard prompts.

4.2 Generation Quality and Throughput

Table 1 compares compact BF16 models [47–49], quantized and low-bit AR models, and learned draft–verify systems with BitMTP on H20. The matched SFT AR routes establish the acceleration obtained beyond packed low-bit execution. At $D{=}8$, hybrid decoding achieves five-task mean speedups of $2.2\times$ on BitNet, $2.4\times$ on Bonsai-1.7B, and $2.1\times$ on Bonsai-4B. The Bonsai mean scores remain close to SFT: 47.91 versus 47.36 and 63.43 versus 63.55. The compact-model comparisons further place this quality in context. With hybrid decoding, BitMTP–Bonsai-1.7B approaches BF16 Qwen2.5-1.5B's 49.23 and exceeds its AWQ variant's 41.79, while BitMTP–Bonsai-4B exceeds Llama-3.2-3B's 57.41 and Qwen3-1.7B's 58.23. BitMTP therefore supports multi-token acceleration on both smaller deployment models and stronger backbones.

The EAGLE-3 [21] comparison examines learned speculative drafting with a packed low-bit target. Bonsai+EAGLE-3 reports 566.23 tokens/s at 47.65 mean quality, whereas BitMTP hybrid decoding at $D{=}8$ reaches 1110.65 tokens/s at 47.91. BitMTP combines block proposals and AR verification in one low-bit backbone rather than adding a BF16 drafter. Hybrid decoding exceeds confidence-only decoding in both mean quality and throughput across all five model–block-size configurations in Table 1. On Bonsai-1.7B at $D{=}8$, hybrid decoding and speculative-only decoding score 47.91 and 47.89, while hybrid throughput increases from 1028.13 to 1110.65 tokens/s. BitNet offers a different choice when quality retention is prioritized: $D{=}4$ speculative-only decoding scores 45.17 against SFT's 45.95 with $1.7\times$ throughput, whereas $D{=}8$ hybrid decoding reaches $2.2\times$ at 43.25. On Bonsai-4B, speculative-only decoding achieves a mean score of 64.13, compared with 63.55 for SFT, while retaining a $1.9\times$ throughput speedup.

Increasing D from 4 to 8 raises hybrid mean throughput from 722.12 to 805.57 tokens/s on BitNet and from 952.18 to 1110.65 on Bonsai-1.7B. Useful progress depends on the committed prefix rather than the proposed width alone. For Bonsai-1.7B hybrid decoding at $D{=}8$, BitMTP obtains $2.7\times$ and

Table 1: Generation quality (Sc., %) and decode throughput (Rate, tokens/s) on NVIDIA H20. Dark-red values in parentheses give speedups over AR baselines. Average is the unweighted five-task mean. W/A denotes weight/activation precision. Conf., Spec., and Hybr. denote confidence-only, speculative-only, and hybrid decoding, respectively.

Model	W/A	Type	GSM8K		MATH-500		IFEval		HumanEval+		MBPP+		Average	
			Sc.	Rate	Sc.	Rate	Sc.	Rate	Sc.	Rate	Sc.	Rate	Sc.	Rate
Compact, quantized, and speculative baselines														
MiniCPM-2B	BF16/BF16	AR	47.84	273.83	11.00	301.18	43.65	306.92	40.24	290.12	43.65	252.08	37.28	284.82
Qwen2.5-1.5B	BF16/BF16	AR	57.54	441.38	49.00	470.88	51.08	467.00	37.20	439.50	51.32	395.61	49.23	442.88
Llama-3.2-3B	BF16/BF16	AR	66.79	274.71	45.80	303.84	78.90	303.35	44.51	284.95	51.06	256.08	57.41	284.59
Qwen3-1.7B	BF16/BF16	AR	68.16	452.38	67.40	454.19	75.30	457.90	37.20	456.47	43.12	464.42	58.23	457.07
Qwen2.5-1.5B-AWQ	INT4/FP16	AR	50.49	266.26	37.80	268.65	47.12	270.05	29.88	268.86	43.65	267.39	41.79	268.24
Llama3-8B-1.58†	1.58/BF16	AR	3.94	13.77	1.00	13.79	30.70	13.83	0.00	14.06	0.79	14.03	7.29	13.90
Falcon3-7B-1.58bit†	1.58/BF16	AR	1.67	14.99	18.40	14.98	43.76	13.84	35.98	15.24	23.02	15.71	24.56	14.95
BitCPM4-1B†	1.58/BF16	AR	41.85	26.66	29.80	22.81	72.18	29.32	17.07	27.95	52.12	27.88	42.60	26.92
BitNet b1.58-2B family: packed W2A8														
BitNet b1.58-2B	1.58/INT8	AR	44.28	363.53	40.80	373.12	62.11	360.55	34.76	355.99	34.92	378.73	43.37	366.38
BitNet b1.58-2B-SFT	1.58/INT8	AR	58.00	364.97	35.80	364.97	55.76	368.73	40.24	393.65	39.95	374.78	45.95	373.42
BitMTP–BitNet b1.58-2B (D=4)	1.58/INT8	MTP (Conf.)	61.11	691.59 (1.9×)	32.80	614.13 (1.7×)	54.80	467.65 (1.3×)	35.37	698.26 (1.8×)	32.54	682.60 (1.8×)	43.32	630.84 (1.7×)
		MTP (Spec.)	65.05	662.78 (1.8×)	37.20	640.38 (1.8×)	54.44	535.81 (1.5×)	34.76	658.55 (1.7×)	34.39	671.77 (1.8×)	45.17	633.86 (1.7×)
		MTP (Hybr.)	63.08	761.42 (2.1×)	32.60	707.11 (1.9×)	55.64	580.35 (1.6×)	33.54	768.71 (2.0×)	34.92	793.01 (2.1×)	43.95	722.12 (1.9×)
BitMTP–BitNet b1.58-2B (D=8)	1.58/INT8	MTP (Conf.)	56.48	647.49 (1.8×)	29.60	539.27 (1.5×)	53.36	409.84 (1.1×)	32.32	691.87 (1.8×)	31.75	728.68 (1.9×)	40.70	603.43 (1.6×)
		MTP (Spec.)	60.35	718.60 (2.0×)	32.60	698.14 (1.9×)	55.28	503.21 (1.4×)	34.15	772.56 (2.0×)	32.80	790.60 (2.1×)	43.04	696.62 (1.9×)
		MTP (Hybr.)	60.35	819.42 (2.2×)	33.60	776.99 (2.1×)	54.56	543.59 (1.5×)	34.15	923.63 (2.3×)	33.60	964.20 (2.6×)	43.25	805.57 (2.2×)
Bonsai-1.7B family: packed W2A8														
Bonsai-1.7B	1.58/INT8	AR	64.14	469.36	46.20	449.62	78.78	465.63	41.46	475.39	20.90	473.44	50.30	466.69
Bonsai-1.7B-SFT	1.58/INT8	AR	59.67	472.99	41.20	447.08	68.59	473.98	31.10	477.83	36.24	474.42	47.36	469.26
Bonsai-1.7B + EAGLE-3	W2A8+BF16	MTP	58.98	515.55 (1.1×)	40.80	537.35 (1.2×)	68.71	547.04 (1.2×)	33.54	600.11 (1.3×)	36.24	631.10 (1.3×)	47.65	566.23 (1.2×)
BitMTP–Bonsai-1.7B (D=4)	1.58/INT8	MTP (Conf.)	54.89	701.25 (1.5×)	41.80	837.48 (1.9×)	64.75	648.91 (1.4×)	32.93	943.90 (2.0×)	36.51	847.30 (1.8×)	46.17	795.77 (1.7×)
		MTP (Spec.)	57.85	795.80 (1.7×)	44.20	853.38 (1.9×)	66.43	714.93 (1.5×)	32.93	888.55 (1.9×)	38.89	867.85 (1.8×)	48.06	824.10 (1.8×)
		MTP (Hybr.)	57.47	901.35 (1.9×)	42.20	979.79 (2.2×)	65.35	791.36 (1.7×)	32.32	1078.32 (2.3×)	38.89	1010.11 (2.1×)	47.24	952.18 (2.0×)
BitMTP–Bonsai-1.7B (D=8)	1.58/INT8	MTP (Conf.)	54.21	661.80 (1.4×)	39.60	837.24 (1.9×)	66.19	593.09 (1.3×)	34.15	1010.59 (2.1×)	35.71	1040.99 (2.2×)	45.97	828.74 (1.8×)
		MTP (Spec.)	57.70	907.20 (1.9×)	41.60	1051.18 (2.4×)	68.11	760.12 (1.6×)	34.76	1184.89 (2.5×)	37.30	1237.25 (2.6×)	47.89	1028.13 (2.2×)
		MTP (Hybr.)	57.47	919.54 (1.9×)	43.20	1112.68 (2.5×)	68.59	801.42 (1.7×)	33.54	1288.33 (2.7×)	36.77	1431.28 (3.0×)	47.91	1110.65 (2.4×)
Bonsai-4B family: packed W2A8														
Bonsai-4B-SFT	1.58/INT8	AR	79.08	278.04	61.20	273.46	76.62	277.11	52.44	278.66	48.41	279.59	63.55	277.37
BitMTP–Bonsai-4B (D=8)	1.58/INT8	MTP (Conf.)	76.35	347.76 (1.3×)	55.40	473.75 (1.7×)	74.94	281.91 (1.0×)	48.78	545.92 (2.0×)	53.17	509.61 (1.8×)	61.73	431.79 (1.6×)
		MTP (Spec.)	78.54	504.21 (1.8×)	62.20	564.80 (2.1×)	78.30	365.03 (1.3×)	50.00	612.29 (2.2×)	51.59	603.92 (2.2×)	64.13	530.05 (1.9×)
		MTP (Hybr.)	78.17	513.27 (1.8×)	59.60	612.41 (2.2×)	77.34	369.50 (1.3×)	49.39	701.86 (2.5×)	52.65	709.18 (2.5×)	63.43	581.24 (2.1×)

†BF16 reference execution.

3.0× speedups on HumanEval+ and MBPP+, and a 1.7× speedup on IFEval. These runs commit approximately 6.7 and 7.1 tokens per iteration on the code tasks, versus 4.6 on IFEval. This pattern is consistent with amortizing block computation and state updates over more committed tokens, connecting workload-dependent acceleration to the execution state managed by BitMTP. With D=8 hybrid decoding, both Bonsai-1.7B and Bonsai-4B achieve more than twice the five-task mean throughput of their respective SFT baselines while maintaining comparable mean task quality. These results show that BitMTP retains substantial acceleration as the backbone scales from 1.7B to 4B parameters.

Table 2: Five-task quality (Sc., %) and decode throughput (Rate, tokens/s) of hybrid decoding on NVIDIA RTX 3090. Parentheses give speedups over SFT AR; Average is the unweighted task mean.

Configuration	GSM8K		MATH-500		IFEval		HumanEval+		MBPP+		Average	
	Sc.	Rate	Sc.	Rate	Sc.	Rate	Sc.	Rate	Sc.	Rate	Sc.	Rate
BitNet b1.58-2B-SFT	58.38	275.54	35.00	281.68	56.35	277.36	39.02	284.19	40.74	282.94	45.90	280.34
BitMTP (D=4)	62.77	510.62 (**1.9**×)	35.00	492.56 (**1.7**×)	54.08	392.57 (**1.4**×)	35.37	572.04 (**2.0**×)	33.60	553.23 (**2.0**×)	44.16	504.20 (**1.8**×)
BitMTP (D=8)	59.89	524.16 (**1.9**×)	33.00	515.08 (**1.8**×)	55.76	364.84 (**1.3**×)	35.37	661.16 (**2.3**×)	33.33	626.82 (**2.2**×)	43.47	538.41 (**1.9**×)
Bonsai-1.7B-SFT	61.18	244.51	42.00	245.31	68.94	244.81	35.37	245.08	36.51	244.64	48.80	244.87
BitMTP (D=4)	57.54	430.60 (**1.8**×)	42.60	472.35 (**1.9**×)	67.39	376.04 (**1.5**×)	34.15	516.57 (**2.1**×)	37.57	474.59 (**1.9**×)	47.85	454.03 (**1.9**×)
BitMTP (D=8)	58.00	416.82 (**1.7**×)	42.40	507.39 (**2.1**×)	66.91	346.49 (**1.4**×)	31.71	580.83 (**2.4**×)	37.83	639.76 (**2.6**×)	47.37	498.26 (**2.0**×)

Table 3: MBPP+ decode throughput (tokens/s) on Jetson Orin. Parentheses give speedups over SFT AR; decoding-policy abbreviations follow Table 1.

Backbone	SFT AR	BitMTP (D=4)			BitMTP (D=8)		
		Conf.	Spec.	Hybr.	Conf.	Spec.	Hybr.
BitNet b1.58-2B	19.35 (**1.0**×)	35.38 (**1.8**×)	34.38 (**1.8**×)	40.92 (**2.1**×)	44.74 (**2.3**×)	54.55 (**2.8**×)	61.76 (**3.2**×)
Bonsai-1.7B	15.95 (**1.0**×)	30.80 (**1.9**×)	28.83 (**1.8**×)	36.20 (**2.3**×)	40.42 (**2.5**×)	42.47 (**2.7**×)	52.05 (**3.3**×)

Table 4: Eleven-task model capability scores (%). Avg. denotes the unweighted mean; TQA is TruthfulQA-mc2. W/A denotes weight/activation precision. TriviaQA scores generated answers; the remaining tasks use likelihood scoring.

Model	W/A	MMLU	Wino	Hella	ARC-C	ARC-E	BoolQ	OBQA	PIQA	Trivia	CSQA	TQA	Avg.
MiniCPM-2B	BF16/BF16	52.4	65.5	71.4	43.8	68.9	77.6	42.6	76.1	34.2	73.1	43.1	59.0
Qwen2.5-1.5B	BF16/BF16	60.2	63.4	68.3	46.6	75.9	78.1	40.4	75.9	39.0	76.5	46.7	61.0
Llama-3.2-3B	BF16/BF16	60.7	68.6	71.6	46.1	71.0	78.8	39.2	76.9	58.6	72.5	51.5	63.2
Qwen2.5-1.5B-AWQ	INT4/FP16	57.5	63.5	66.4	44.3	72.5	74.4	40.4	75.6	33.9	74.0	45.4	58.9
Llama3-8B-1.58†	1.58/BF16	35.2	59.2	68.7	43.4	69.8	68.6	39.4	75.9	20.0	30.0	39.1	49.9
Falcon3-7B-1.58bit†	1.58/BF16	43.3	60.1	59.5	37.6	64.8	72.3	38.6	72.8	0.0	67.3	43.2	50.9
BitCPM4-1B†	1.58/BF16	55.2	55.8	52.6	38.9	63.0	70.4	36.6	69.1	15.4	55.0	42.5	50.4
BitNet b1.58-2B family													
BitNet b1.58-2B	1.58/INT8	53.0	72.1	67.6	48.1	73.7	79.2	40.0	76.6	33.7	71.8	46.4	60.2
BitNet b1.58-2B-SFT	1.58/INT8	53.2	69.6	67.8	50.6	76.1	79.5	42.4	78.1	33.6	69.6	39.8	60.0
BitMTP (D=4)	1.58/INT8	52.9	70.2	67.2	48.8	75.1	79.1	41.8	78.0	33.0	70.3	41.7	59.8
BitMTP (D=8)	1.58/INT8	52.0	70.0	66.0	48.6	76.5	78.7	39.8	77.2	29.8	69.4	42.8	59.1
Bonsai-1.7B family													
Bonsai-1.7B	1.58/INT8	49.2	59.1	52.4	38.5	65.2	72.4	33.8	70.1	19.8	63.4	46.9	51.9
Bonsai-1.7B-SFT	1.58/INT8	46.1	58.0	53.8	37.8	64.1	71.7	33.6	69.4	19.2	60.4	43.4	50.7
BitMTP (D=4)	1.58/INT8	43.0	55.7	52.3	35.6	64.5	70.4	32.2	70.6	17.7	56.7	42.7	49.2
BitMTP (D=8)	1.58/INT8	44.0	56.7	52.3	36.2	63.8	74.4	33.8	70.2	17.5	59.9	41.8	50.0

†BF16 reference execution.

Tables 2 and 3 extend the evaluation to consumer and edge GPUs. On RTX 3090, D=8 hybrid decoding achieves five-task mean speedups of $1.9\times$ on BitNet and $2.0\times$ on Bonsai-1.7B. On Jetson Orin's MBPP+ workload, BitNet throughput increases from 19.35 to 61.76 tokens/s and Bonsai from 15.95 to 52.05, giving $3.2\times$ and $3.3\times$ speedups. All three policies accelerate MBPP+ over SFT AR on Orin, with hybrid decoding the fastest at both block sizes. Together, these results demonstrate effective multi-token acceleration across server, consumer, and edge GPUs, showing that the benefits of BitMTP extend beyond a single hardware class.

4.3 Model Capability Retention

Table 4 compares BitMTP with the SFT counterpart of each backbone to assess retention of knowledge, reasoning, and commonsense capabilities beyond the generation benchmarks. Ten tasks assess the adapted backbone through causal likelihood scoring, without invoking multi-token

Table 5: BitNet hybrid-decoding configurations at $D \in \{4, 8\}$. Rates are decode tokens/s; Avg. is the unweighted task mean. Parentheses give speedups over SFT AR; HE+ denotes HumanEval+.

Runtime configuration	D=4 (tokens/s)						D=8 (tokens/s)					
	GSM8K	MATH-500	IFEval	HE+	MBPP+	Avg.	GSM8K	MATH-500	IFEval	HE+	MBPP+	Avg.
SFT AR	365.0	365.0	368.7	393.7	374.8	373.4 (1.0×)	365.0	365.0	368.7	393.7	374.8	373.4 (1.0×)
Generic	158.0	133.0	103.8	157.5	162.9	143.0 (0.4×)	268.9	215.9	144.2	290.5	301.3	244.2 (0.7×)
+ M1	724.2	658.2	515.1	739.2	758.0	678.9 (1.8×)	759.1	663.7	429.6	793.7	840.6	697.4 (1.9×)
+ M1+M2	735.2	664.8	529.1	746.0	772.5	689.5 (1.8×)	784.5	684.3	452.4	825.3	879.6	725.2 (1.9×)
+ M1+M2+M3 (Full)	761.4	707.1	580.4	768.7	793.0	722.1 (1.9×)	819.4	777.0	543.6	923.6	964.2	805.6 (2.2×)

acceptance policies; TriviaQA evaluates generated answers by exact match. Relative to SFT, BitNet's eleven-task means remain close at 59.8 and 59.1 for D=4 and D=8, compared with 60.0. Bonsai-1.7B scores 49.2 and 50.0, compared with 50.7 for SFT. The block-size ordering differs across backbones: D=4 has the higher eleven-task mean on BitNet, whereas D=8 scores higher on Bonsai-1.7B. On both backbones, BitMTP maintains eleven-task mean scores close to SFT, with differences of 0.2–1.5 percentage points.

BitNet at D=4 scores higher than SFT on WinoGrande and TruthfulQA-mc2, while Bonsai-1.7B at D=8 scores 74.4 versus 71.7 on BoolQ but lower on MMLU and HellaSwag. The aggregate scores thus reflect a task-dependent pattern of retained strengths, gains, and reductions. Relative to compact-model baselines, BitNet's MTP means remain near MiniCPM-2B's 59.0 and Qwen2.5-1.5B-AWQ's 58.9, while BF16 Qwen2.5-1.5B and Llama-3.2-3B score higher at 61.0 and 63.2. Together with Section 4.2, these results provide complementary evidence for BitMTP's generation acceleration and broad capability retention relative to SFT.

4.4 Runtime Analysis

Decode efficiency. Table 5 compares different runtime configurations. The Generic configuration reaches a five-task mean throughput of 143.0 tokens/s at D=4 and 244.2 tokens/s at D=8. Both rates are below the SFT AR baseline of 373.42 tokens/s. The Full configuration reaches 722.12 tokens/s at D=4 and 805.57 tokens/s at D=8. It achieves the highest throughput on all five tasks at both block sizes. At D=8, its task-level speedups range from 1.5× on IFEval to 2.6× on MBPP+. These gains span mathematical reasoning, instruction following, and code generation.

Acceptance and runtime cost. In the forced-rejection test reported in **Appendix C**, each round performs block drafting and verification but commits at most one verifier-origin token, resulting in throughput below AR. In contrast, evaluated Bonsai-1.7B hybrid decoding at D=8 commits 4.6–7.1 tokens per iteration across five tasks. Multi-token commitment amortizes block execution and verification. **Appendix D** quantifies the memory and initialization costs of deploying the full runtime. The 0.5–1.6 MiB shared-KV increment on H20 supports history reuse across replay graphs, while Orin's one-time startup overhead is amortized in persistent serving. Therefore, the proposed BitMTP translates multi-token commitment into decode speedups while sharing causal history across replay states.

5 Conclusion and Limitation

BitMTP enabled efficient multi-token generation on ultra-low-bit LLM backbones through joint NTP+MTP training, three decoding policies, and accepted-prefix-aligned runtime execution. The runtime combined width-specialized operators, shared-KV graph replay, and verifier scheduling. At D=8 on H20, hybrid decoding reached five-task mean speedups of 2.2× on BitNet b1.58-2B and 2.1–2.4× on 1.7B–4B Bonsai models at comparable task quality. BitMTP reached up to 2.0× in five-task mean throughput on RTX 3090 and 3.3× on MBPP+ on Jetson Orin. These results demonstrated effective low-bit multi-token acceleration across ternary backbones, model scales, and GPU types.

There are some limitations of BitMTP. First, while BitMTP has been evaluated on 1.7B–4B backbones, its quality–throughput tradeoffs on larger models and longer contexts still need validation. Second, the current runtime relies on CUDA-specific low-bit kernels and CUDA Graphs and cannot run directly on non-CUDA accelerators. We will investigate these problems in further work.

Acknowledgments and Disclosure of Funding

This work is supported by the PolyU-OPPO Joint Innovative Research Center. The authors declare no competing interests.

References

[1] Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yineng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, et al. Flashinfer: Efficient and customizable attention engine for llm inference serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

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

[3] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, et al. Specinfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 932–949, 2024.

[4] Hongyu Wang, Shuming Ma, Lingxiao Ma, Lei Wang, Wenhui Wang, Li Dong, Shaohan Huang, Huaijie Wang, Jilong Xue, Ruiping Wang, et al. Bitnet: 1-bit pre-training for large language models. *Journal of Machine Learning Research*, 26(125):1–29, 2025.

[5] Shuming Ma, Hongyu Wang, Shaohan Huang, Xingxing Zhang, Ying Hu, Ting Song, Yan Xia, and Furu Wei. Bitnet b1. 58 2b4t technical report. *arXiv preprint arXiv:2504.12285*, 2025.

[6] MiniCPM Team, Chaojun Xiao, Yuxuan Li, Xu Han, Yuzhuo Bai, Jie Cai, Haotian Chen, Wentong Chen, Xin Cong, Ganqu Cui, et al. Minicpm4: Ultra-efficient llms on end devices. *arXiv preprint arXiv:2506.07900*, 2025.

[7] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. *arXiv preprint arXiv:2401.15077*, 2024.

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

[9] Fangcheng Liu, Yehui Tang, Zhenhua Liu, Yunsheng Ni, Duyu Tang, Kai Han, and Yunhe Wang. Kangaroo: Lossless self-speculative decoding for accelerating llms via double early exiting. *Advances in Neural Information Processing Systems*, 37:11946–11965, 2024.

[10] Yangzhou Liu, Yue Cao, Hao Li, Gen Luo, Zhe Chen, Weiyun Wang, Xiaobo Liang, Biqing Qi, Lijun Wu, Changyao Tian, et al. Sequential diffusion language models. *arXiv preprint arXiv:2509.24007*, 2025.

[11] 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)*, pages 11263–11282, 2024.

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

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

[14] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pages 38087–38099. PMLR, 2023.

[15] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of machine learning and systems*, 6:87–100, 2024.

[16] Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. Llm-qat: Data-free quantization aware training for large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 467–484, 2024.

[17] PrismML. Ternary Bonsai 8B: Ternary (1.58-bit) language models at 8B, 4B, and 1.7B scale. Technical report, PrismML, April 2026. URL `https://github.com/PrismML-Eng/Bonsai-demo/blob/main/ternary-bonsai-8b-whitepaper.pdf`.

[18] Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit llms: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

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

[20] 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*, pages 7421–7432, 2024.

[21] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle-3: Scaling up inference acceleration of large language models via training-time test. *arXiv preprint arXiv:2503.01840*, 2025.

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

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

[24] John Kirchenbauer, Abhimanyu Hans, Brian Bartoldson, Micah Goldblum, Ashwinee Panda, and Tom Goldstein. Multi-token prediction via self-distillation. *arXiv preprint arXiv:2602.06019*, 2026.

[25] Juhyeong Kim, Sangyeon Yu, Gyunyeop Kim, and Sangwoo Kang. Fractalllm: Lossless self-speculative decoding with layer embedded self-compression. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 23666–23673, 2025.

[26] Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. *Advances in Neural Information Processing Systems*, 37:129531–129563, 2024.

[27] Yunlong Hou, Fengzhuo Zhang, Cunxiao Du, Xuan Zhang, Jiachun Pan, Tianyu Pang, Chao Du, Vincent YF Tan, and Zhuoran Yang. Banditspec: Adaptive speculative decoding via bandit algorithms. *arXiv preprint arXiv:2505.15141*, 2025.

[28] Jian Chen, Yesheng Liang, and Zhijian Liu. Dflash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026.

[29] Juntao Zhao, Wenhao Lu, Sheng Wang, Lingpeng Kong, and Chuan Wu. Qspec: Speculative decoding with complementary quantization schemes. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4779–4795, 2025.

[30] Evangelos Georganas, Dhiraj Kalamkar, Alexander Kozlov, and Alexander Heinecke. Ml-specqd: Multi-level speculative decoding with quantized drafts. *arXiv preprint arXiv:2503.13565*, 2025.

[31] Rishabh Tiwari, Haocheng Xi, Aditya Tomar, Coleman Hooper, Sehoon Kim, Maxwell Horton, Mahyar Najibi, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Quantspec: Self-speculative decoding with hierarchical quantized kv cache. *arXiv preprint arXiv:2502.10424*, 2025.

[32] Jinheng Wang, Hansong Zhou, Ting Song, Shijie Cao, Yan Xia, Ting Cao, Jianyu Wei, Shuming Ma, Hongyu Wang, and Furu Wei. Bitnet. cpp: Efficient edge inference for ternary llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9305–9322, 2025.

[33] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209, 2024.

[34] Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. Qserve: W4a8kv4 quantization and system co-design for efficient llm serving. *Proceedings of Machine Learning and Systems*, 7, 2025.

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

[36] Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. *Advances in Neural Information Processing Systems*, 37:68658–68685, 2024.

[37] Ke Hong, Guohao Dai, Jiaming Xu, Qiuli Mao, Xiuhong Li, Jun Liu, Kangdi Chen, Yuhan Dong, and Yu Wang. Flashdecoding++: Faster large language model inference with asynchronization, flat gemm optimization, and heuristics. *Proceedings of Machine Learning and Systems*, 6:148–161, 2024.

[38] Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. vattention: Dynamic memory management for serving llms without pagedattention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pages 1133–1150, 2025.

[39] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. InfiniGen: Efficient generative inference of large language models with dynamic KV cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 155–172, 2024.

[40] Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models. *Advances in Neural Information Processing Systems*, 37:139997–140031, 2024.

[41] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

[42] Wen Cheng, Shichen Dong, Jiayu Qin, and Wei Wang. Qaq: Quality adaptive quantization for llm kv cache. In *2025 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, pages 2563–2571. IEEE, 2025.

[43] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. Llumnix: Dynamic scheduling for large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pages 173–191, 2024.

[44] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pages 117–134, 2024.

[45] 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.

[46] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pages 193–210, 2024.

[47] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. Minicpm: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

[48] Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL `https://qwenlm.github.io/blog/qwen2.5/`.

[49] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Appendix

In this appendix, we provide the following materials:

- **A.** Method and runtime details, including one hybrid decoding iteration, verifier and KV-state transitions, and replay-correctness requirements (referring to Section 3 in the main paper).
- **B.** Training data and cost (referring to Section 4.1 in the main paper).
- **C.** Forced-rejection analysis for the runtime configuration study (referring to Section 4.4 in the main paper).
- **D.** GPU-resident model and runtime memory, graph-state overhead, and Jetson Orin cold-start measurements (referring to Section 4.4 in the main paper).

A Method and Runtime Details

Algorithm 1 summarizes one hybrid decoding iteration. The draft replay computes causal KV for the pending suffix while proposing a new block, and the runtime copies the pending-token KV rows into history. Hybrid decoding commits a sufficiently long confidence prefix; otherwise, an AR-causal verifier processes $[\text{anchor}, d_1, \ldots, d_D]$ at width $D{+}1$ and determines the accepted prefix. Its additional token corrects the first rejected position or serves as a bonus after full acceptance. Confidence-only decoding uses the confidence branch, while speculative-only decoding always uses the verifier branch.

Algorithm 1 Hybrid decoding state update.

Require: state $q = (p, \ell, u)$ with $u \neq []$, block size D, and controls (τ, m)
set $n \leftarrow |u|$ and select $s_{\mathrm{d}} = (M, \sigma, r_{\mathrm{d}})$ with $P = M - D \geq n$
form $x_{\mathrm{d}} = [u, \mathrm{PAD}^{P-n}, \mathrm{last}(u), m_{\mathrm{mask}}^{D-1}]$
replay the draft graph on x_{d} and obtain $d_{1:D}$ from the final D output rows
materialize only the n pending KV rows and set $\bar{\ell} \leftarrow \ell + n$
compute h on the GPU using Eq. 4
select $s_{\mathrm{v}} = (D{+}1, \sigma, r_{\mathrm{v}})$ and prepare $[\mathrm{last}(u), d_{1:D}]$
prepare verifier positions and the AR-causal mask on the GPU stream
obtain h on the host
if $h \geq m$ **then**
    set $\hat{a} \leftarrow h$
    append $d_{1:\hat{a}}$ to p
    set $u' \leftarrow d_{1:\hat{a}}$ and $\ell' \leftarrow \bar{\ell}$
else
    replay the verifier graph for s_{v} and obtain $\tilde{d}_{1:D+1}$
    set $\hat{a}$ to the verifier-accepted prefix length
    set $b \leftarrow \tilde{d}_{\hat{a}+1}$
    promote the $\hat{a}$ accepted draft KV rows, excluding the anchor
    append $d_{1:\hat{a}} \cdot b$ to p
    set $u' \leftarrow [b]$ and $\ell' \leftarrow \bar{\ell} + \hat{a}$
end if
return $q' = (p, \ell', u')$

After a confidence transition, the committed drafts remain pending in the updated state until a later causal replay computes their KV from the emitted token ids. A verifier transition instead promotes the accepted causal rows and leaves only the verifier-origin token pending. Both branches preserve Eq. 1. In the verifier branch, full acceptance appends the $(D{+}1)$-st verifier token as a bonus, whereas rejection at the first draft position emits only the verifier correction.

Superscripts on PAD and m_{mask} denote repeated input slots. The draft anchor repeats the final pending token and its position; block queries hide its original pending copy. Verifier positions start at $\ell - 1$, and its mask hides the anchor's history copy. Padding, mask, and rejected draft rows do not enter committed causal history. Each reachable pending length n has a captured draft width $M \geq n + D$; verification uses width $D{+}1$. Graphs of different widths and roles share B_σ, with history migration only when capacity grows. These choices preserve the prefix-length relation in Eq. 1.

Table 6: Composition of the processed training mixture used in the training-cost study.

Source	**Examples**	**Domain**
ScaleQuest-Math	1,003,467	Math
SmolTalk	1,038,618	General
OPC-SFT-Stage2	435,930	Code
Tulu-3-SFT-Mixture	923,278	General
SciRIFF	70,520	Science
Table-GPT	13,214	Table
OpenCodeInstruct	17,494	Code
Reasoning-SFT-IF	41,413	Instruction following
Total	**3,543,934**	N/A

Table 7: One-epoch training cost on Bonsai-1.7B using eight NVIDIA H20 GPUs. All configurations use the same corpus, global batch size, and optimizer-step count. Relative cost uses the one-epoch SFT run as its baseline.

Configuration	Wall time (h)	GPU-hours	Relative cost
SFT	71.29	570.3	1.00×
BitMTP (D=4)	103.10	824.8	1.45×
BitMTP (D=8)	112.88	903.0	1.58×

Table 8: Forced-rejection stress test on H20. Decode gives five-task mean throughput (tokens/s); dark-red values in parentheses show speedups over SFT AR. Accepted and Com. report per-iteration means of accepted draft tokens and committed output tokens.

Backbone	**SFT AR Decode**	**Forced rejection, D=4**			**Forced rejection, D=8**		
		Decode	Accepted	Com.	Decode	Accepted	Com.
BitNet b1.58-2B	373.42 **(1.0×)**	142.91 **(0.4×)**	0	0.9975	103.40 **(0.3×)**	0	0.9974
Bonsai-1.7B	469.26 **(1.0×)**	188.57 **(0.4×)**	0	0.9978	160.28 **(0.3×)**	0	0.9978

B Training Data and Cost

Table 6 lists the eight training sources, covering mathematics, general dialogue, code, science, tables, and instruction following. For each backbone, SFT and joint NTP+MTP training use the same processed mixture drawn from these sources, comprising approximately 3.5 million examples.

Table 7 reports one-epoch training cost on Bonsai-1.7B using eight H20 GPUs. With the corpus and optimizer-step count fixed, joint NTP+MTP training uses $1.45\times$/$1.58\times$ the SFT GPU-hours at D=4/8, quantifying the additional cost of joint AR and block-prediction supervision.

C Additional Experimental Analyses

Forced-rejection stress test. The stress test in Table 8 forces zero acceptance in each draft–verify round, paying for D drafts and width-$(D{+}1)$ verification while committing at most one verifier-origin token. The result is below AR, with lower throughput at D=8 than at D=4 on both backbones. Individual zero-acceptance rounds can occur normally, but this all-reject stress test is an artificial worst case. Evaluated Bonsai-1.7B hybrid workloads at D=8 commit 4.6–7.1 tokens per iteration, amortizing drafting and verification to support the measured low-bit generation speedups.

D Runtime Memory and Startup Cost

Tables 9 and 10 quantify memory and startup costs at a 4,096-token segment. SFT AR captures one graph and BitMTP captures seven graphs over shared causal KV. Model totals include all resident weight representations and the embedding and output-projection storage.

Table 9: H20 model and runtime memory at a 4,096-token segment, in MiB. Model total includes Emb./head. Static contains per-graph token, position, and mask buffers. Other allocated is CUDA-allocated memory after graph capture minus Model total, Shared KV, and Static. Peak is the median of three runs' maximum device-memory readings across the recorded stages.

Configuration	Graphs	Emb./head	BF16 prefill	Packed decode	Model total	Shared KV	Static	Other allocated	Peak
BitNet b1.58-2B									
SFT AR	1	2505.0	6563.3	881.6	9950.0	300.1	0.5	132.4	11028
BitMTP (D=4)	7	2818.1	6563.3	881.6	10263.1	300.6	13.7	326.8	11714
BitMTP (D=8)	7	2818.1	6563.3	881.6	10263.1	301.2	26.9	327.6	11742
Bonsai-1.7B									
SFT AR	1	671.3	0.0	651.2	1322.5	448.1	0.6	0.0	5118
BitMTP (D=4)	7	967.5	0.0	651.2	1618.8	448.9	16.1	165.1	5656
BitMTP (D=8)	7	967.5	0.0	651.2	1618.8	449.8	31.5	165.1	5656

Table 10: Jetson Orin memory and cold-start cost at a 4,096-token segment. Memory is in MiB. Capture is the allocator increase from KV allocation to ready-to-serve. Peak reserved is the first-request CUDA allocator peak; cold start uses an empty compiler cache.

Configuration	Graphs	Shared KV	Capture	Peak reserved	Cold start (s)
BitNet b1.58-2B-SFT AR	1	301.2	16.7	11312	14.34
BitMTP–BitNet (D=4)	7	300.7	79.2	11716	17.30
BitMTP–BitNet (D=8)	7	301.2	93.4	11730	17.92
Bonsai-1.7B-SFT AR	1	448.1	16.8	5486	13.41
BitMTP–Bonsai (D=4)	7	449.0	81.6	5876	16.52
BitMTP–Bonsai (D=8)	7	449.9	97.9	5960	16.53

H20 shared KV increases by only 0.5–1.6 MiB from one to seven graphs, reusing causal history rather than duplicating full caches. Sampled peak device memory is 686–714 MiB higher on BitNet and 538 MiB higher on Bonsai-1.7B than on SFT AR.

Orin's first-request peak reserved memory increases by 404–418 MiB on BitNet and 390–474 MiB on Bonsai-1.7B, with cold-start increments of 2.96–3.58 s and 3.11–3.12 s, respectively. This one-time per-process cost is amortized over requests, supporting persistent multi-token generation through shared history and replayable execution states.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Yes, the main claims in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitations in the final section (Conclusion and Limitation).

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not include theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Yes, the paper fully discloses all necessary information to reproduce the main experimental results, supporting the main claims and conclusions.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: All training and testing data are publicly available. We will release the codes and model if the paper is accepted.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Yes, the paper provides all necessary training and test details, including data splits, hyperparameters, and optimizer type, to fully understand the results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: We do not report error bars.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We detail the type of compute workers, memory, and execution time in the experiment section.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Yes, the research in the paper fully conforms to the NeurIPS Code of Ethics in every respect.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Yes, we analyze the potential social impact in the final section.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes, we clearly indicate the baseline methods and training/testing data used in the paper. Their licenses permit use within academic scope.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Codes and model will be released if the paper is accepted.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowd sourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: Language models are the object of study, rather than an external tool used as a non-standard component of the research methodology.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.