TAURUS: ACCELERATING PRIVATE LLM INFERENCE VIA HOMOMORPHIC SPECULATIVE DECODING

Anonymous authors
Paper under double-blind review

ABSTRACT

Cloud-hosted large language model (LLM) inference raises privacy concerns, since sensitive user inputs are exposed to the service provider. Fully homomorphic encryption (FHE) offers a promising solution by enabling inference directly over encrypted inputs. However, its substantial computational overhead remains a major obstacle to practical deployment. This overhead is particularly severe in autoregressive decoding, which generates only one token at a time, while FHE is more efficient when many values are packed and processed together. Speculative decoding instead processes multiple tokens in one target forward, creating an opportunity to better utilize FHE's packing ability. However, its speculation-specific operations, including sampling, verification, and commit, are expensive to realize under FHE. In this paper, we propose Taurus, the first framework for fully homomorphic speculative decoding. Taurus packs many speculative candidate tokens together to share expensive FHE computation across multiple token states. To be more efficient and accurate, Taurus develops a homomorphic sampling protocol that avoids scanning the entire vocabulary with encrypted comparisons. Taurus also proposes a private speculative verification protocol that avoids encrypted division and data-dependent branching, together with FIXEDTRACECOMMIT, which writes accepted tokens and KV states without revealing the accepted path. Together, these protocols enable efficient speculative decoding entirely on the server, reducing decoder latency per committed token while preserving privacy for both the client and the server. We implement Taurus and conduct experiments on six open-weight models with 1.7B to 8B parameters. On eight NVIDIA RTX 4090 GPUs, Taurus achieves a $1.80\times$–$3.02\times$ speedup over CacheMir in end-to-end decoder latency per committed token. Our implementation is publicly available as open source.

1 INTRODUCTION

Driven by the growing use of cloud-hosted large language models (LLMs), increasingly sensitive personal, enterprise, and proprietary data is now processed by remote model providers. In standard deployments, user prompts and intermediate activations are exposed to the server, raising concerns about data leakage, unauthorized retention, and secondary use. Fully homomorphic encryption (FHE) offers a promising solution by enabling computation directly on encrypted data, protecting both client and server. As illustrated in Figure 1(a), the client encrypts the prompt, the server evaluates the model directly on ciphertexts, and the client later decrypts the returned response. Unlike secure multiparty computation, this setting avoids heavy client-side computation during inference. However, despite recent progress in homomorphic neural-network evaluation, generative LLM inference under FHE remains extremely expensive, especially during autoregressive decoding, where a deep model is repeatedly evaluated to generate one token at a time.

Speculative decoding offers a natural way to amortize this sequential cost. A lightweight draft model first proposes several future tokens, and the target model then verifies them together in one target forward. When draft tokens are accepted, one target forward can produce several output tokens without changing the target distribution (Leviathan et al., 2023; Chen et al., 2023). Building on this idea, tree-based methods evaluate several candidate continuations in parallel and further increase the number of tokens produced by one target forward (Miao et al., 2024; Cai et al., 2024). Such parallelism is particularly attractive under FHE, where every target forward is far more expensive

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.

than in plaintext inference. As shown in Figure 1(b), FHE evaluates linear and nonlinear operations over values packed into these ciphertext slots.

Most prior FHE Transformer systems focus on full-sequence workloads such as prefill, where many token states naturally exist at the same time (Moon et al., 2025; Park et al., 2025; Zhang et al., 2025). However, autoregressive decoding has a very different structure. At every step, the model introduces one new token state but still evaluates the same sequence of linear and nonlinear operations. A few recent systems (Yu et al., 2026; Park et al., 2026) improve ciphertext-slot utilization during decoding through cryptographic packing and related techniques, but the resulting gains remain limited. This is because nonlinear evaluation quickly consumes the available ciphertext levels and triggers ciphertext refresh (*bootstrapping*), one of the most expensive homomorphic operations. Supporting such deep encrypted computation also requires large cryptographic parameters, which provide substantial SIMD capacity. In general configuration, each ciphertext contains 32,768 slots, far more than a single decoding state needs. We therefore face an important mismatch: the expensive nonlinear computation comes with abundant parallel capacity, but conventional autoregressive decoding uses that computation to advance generation by only a single token.

For example, CacheMir fills available ciphertext slots with replicas of the same token state (Yu et al., 2026). This packing amortizes computation for that token, but the model still advances generation by one token after an expensive target forward. In other words, the problem is not simply that FHE computation is expensive. The larger mismatch is that we repeatedly pay this cost while making progress on a single future token.

Motivated by this mismatch, our key observation is that these ciphertext slots can instead hold *different speculative token states*. By placing several candidate continuations together, we can evaluate them in the same target forward and share expensive packed nonlinear computation and bootstrapping across candidate states. As illustrated in Figure 1(c), speculation therefore turns ciphertext capacity that would otherwise hold replicas into computation on possible future tokens.

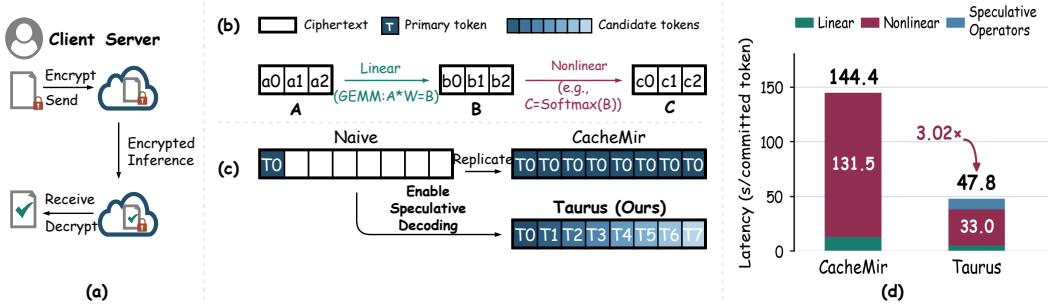

Figure 1: Overview of Taurus. (a) In FHE-based LLM inference, the client encrypts a prompt and later decrypts the response, while the server performs the complete homomorphic computation. (b) Linear and nonlinear operations are evaluated over packed ciphertext slots. (c) CacheMir replicates a single token state, whereas Taurus packs distinct speculative token states for parallel evaluation. (d) For Llama-3.1-8B, Taurus reduces decoder latency per committed token from 144.4 to 47.8 seconds, achieving a 3.02× speedup over CacheMir. Nonlinear operations, including bootstrapping, dominate the decoding latency.

However, realizing speculative decoding entirely under FHE introduces three key challenges. During drafting, each draft forward produces proposal distributions, and we need to sample the token inputs of the next draft depth before the following draft forward can begin. During verification, every rejected candidate changes the target residual, while the usual MSS rule repeatedly normalizes this residual through division. After verification, the accepted tokens and KV states remain scattered across tree nodes, although neither the accepted path nor its length can be revealed. Under FHE, branching on any of these private decisions would expose information through execution or memory-access patterns.

To address these problems, we present TAURUS, the first framework that realizes the complete speculative decoding loop under FHE. For encrypted sampling, we search a sequence of small CDFs

instead of comparing against the entire vocabulary. For verification, we keep the target residual unnormalized and reformulate the MSS acceptance rule so that no encrypted reciprocal is needed. For commit, we use encrypted path masks to gather the accepted tokens and KV states while following the same public execution pattern. Together with a packed forward, these protocols allow us to carry out drafting, target forward, verification, and commit entirely on the server.

Our contributions. First, we identify the mismatch between ciphertext-level parallelism and one-token-at-a-time FHE decoding, and show how speculative states can replace replicated states to turn available ciphertext capacity into future-token computation. Second, we develop a complete FHE speculative-decoding framework with hierarchical sampling, division-free verification, and FIXEDTRACECOMMIT, while preserving the target autoregressive distribution under ideal arithmetic. Finally, we implement Taurus for six models ranging from 1.7B to 8B parameters. Figure 1(d) illustrates the $3.02\times$ speedup for P1, while Figure 6 and Table 7 report $1.80\times$–$3.02\times$ across all six models on eight NVIDIA RTX 4090 GPUs. VERIFYMSS changes 0.0228% of accepted paths relative to FP64 MSS across 442,368 paired tests.

2 PRELIMINARIES

Notation. We use $w[i]$ for entry i, $\odot$ for element-wise multiplication, $\mathrm{ReLU}(w)[i] = \max\{w[i], 0\}$, $\mathbf{1}\{\mathcal{P}\}$ for an indicator, and $\|w\|_1$ for the total mass of nonnegative w.

2.1 AUTOREGRESSIVE AND SPECULATIVE DECODING

Autoregressive decoding generates one token per target forward while reusing a KV cache. Speculative decoding lets a lightweight draft propose several future tokens for one joint target forward (Leviathan et al., 2023; Chen et al., 2023). We use a candidate tree $\mathcal{T}$ rooted at the current token x_0 (Miao et al., 2024; Cai et al., 2024; Chen et al., 2024). A tree mask evaluates all candidates, while SpecInfer's multi-step speculative sampling (MSS) tests siblings in drafting order and updates the target residual after rejections (Miao et al., 2024). The first accepted child continues the path; if none accept, a bonus is sampled from the final residual. A decoding round accepting τ candidates returns $y_1, \ldots, y_\tau, y^\star$. Because both drafting and verification require sampling, we use

$$\mathrm{InvCDF}(w; \omega) = \min \left\{ i : \sum_{k=0}^{i} w[k] \geq \omega \|w\|_1 \right\}, \qquad \omega \in (0, 1]. \tag{1}$$

2.2 CRYPTOGRAPHIC PRIMITIVES AND THREAT MODEL

We combine CKKS for packed approximate arithmetic with TFHE for discrete decisions. CKKS packs $S = N/2$ values per ciphertext and supports SIMD arithmetic, rotations, and plaintext masking (Cheon et al., 2017; 2018; Halevi & Shoup, 2018). For comparisons and short Boolean computations, we switch from CKKS to TFHE for evaluation and return to CKKS without decryption (Chillotti et al., 2020; Lu et al., 2021; Bian et al., 2023). The client holds the prompt, the server holds both models, and both parties are semi-honest (Juvekar et al., 2018; Lu et al., 2025). The server learns neither prompts, outputs, states, nor speculative decisions; the client learns neither model weights nor intermediate outputs. After each decoding round, the client decrypts the output batch and sends a continue/stop signal, revealing the number of decoding rounds. Appendix C.5 specifies the permitted leakage and privacy guarantees.

2.3 RELATED WORK AND GAP

Prior FHE LLM systems mainly optimize full-sequence computation or one-token decoding. NEXUS, THOR, and Powerformer target workloads such as prefill (Zhang et al., 2025; Moon et al., 2025; Park et al., 2025); CacheMir supports autoregressive decoding with an encrypted KV cache (Yu et al., 2026). Each target forward advances generation by only one token. POST (Li et al., 2025b) applies speculative decoding to secure two-party inference, where the client runs the public draft model and participates in verification and resampling. This requires repeated client-side computation and communication and exposes target-side distribution information needed for resampling.

In contrast, Taurus performs drafting, sampling, verification, and commit on the server under FHE, while keeping both speculative decisions and intermediate target-model information encrypted.

3 The Taurus Framework

3.1 Overview

Building on tree-based speculative decoding, we now describe how Taurus uses the parallel capacity of FHE to make progress on several future tokens. Instead of spending one expensive target forward after every newly generated token, we first construct several candidate continuations and then evaluate them together. As illustrated in Figure 2, one Taurus decoding round proceeds in four steps. We alternate draft forwards with encrypted sampling to construct a fixed-shape tree $\mathcal{T}$, perform one target forward over the completed tree, verify an accepted continuation and sample a bonus token, and finally commit the selected tokens and KV states. Algorithm 1 gives the complete fixed-schedule decoding round in Appendix C.

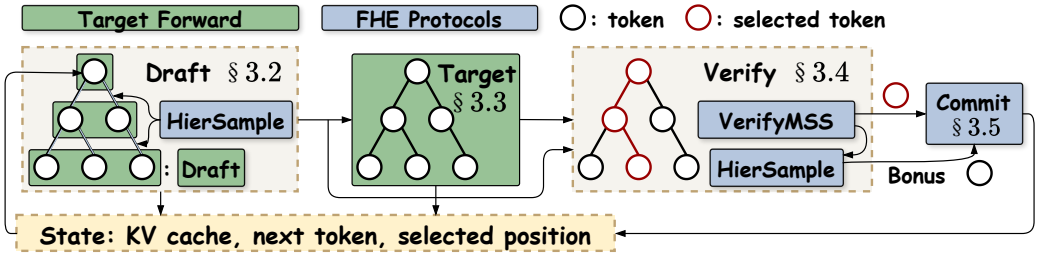

Figure 2: Overview of one Taurus decoding round. We first generate an encrypted draft tree through repeated draft forwards and sampling, then perform one target forward over all candidates, verify the accepted path, and commit the selected tokens and KV states.

With several candidate states available before the target forward, we can place different speculative states in the same CKKS ciphertext and share expensive nonlinear evaluation and bootstrapping across them. However, there are three challenges when directly moving the whole speculative decoding pipeline to FHE. The first challenge arises during drafting. After one draft forward, we obtain proposal distributions for the current depth, but the token inputs of the next depth are not yet known. We need to sample these tokens before the next draft forward can begin, and the same operation appears again when drawing the bonus token. A direct encrypted inverse-CDF search would compare against every vocabulary boundary. A second challenge appears during verification. When we reject a candidate, MSS removes its proposal from the current target residual and normalizes the remaining mass before testing the next sibling. Under FHE, repeating this normalization introduces expensive reciprocal approximation and can propagate numerical error across later decisions. The third challenge arises during commit. After verification, the accepted tokens and KV rows remain at their original tree nodes. Reading the selected nodes directly would reveal the path, while ending the commit at the accepted depth would reveal its length.

To address these challenges, we develop hierarchical encrypted sampling in Section 3.2, pack candidate states for the target forward in Section 3.3, reformulate MSS verification without encrypted division in Section 3.4, and perform FixedTraceCommit with a fixed access pattern in Section 3.5.

3.2 Hierarchical Encrypted Sampling

As discussed above, sampling sits between two consecutive draft forwards. After the draft model evaluates one tree depth, it produces a proposal distribution for every parent node. We sample the child tokens from these proposals, retrieve their embeddings, and use them as the inputs to the next draft forward. Repeating this forward–sampling sequence constructs the candidate tree, and after verification we use the same operation again to draw the bonus token. A straightforward

approach is to apply Equation 1 directly and find the first CDF boundary that crosses a random threshold. Under FHE, this requires 32,000 comparisons for our draft vocabulary and 128,256 or 151,936 comparisons for the native Llama and Qwen3 vocabularies. CKKS can process many boundaries together, but neighboring CDF values can be extremely close, making accurate polynomial comparison expensive (Cheon et al., 2020; Choe et al., 2025). We therefore use TFHE for these comparisons. However, scanning the full vocabulary in TFHE is prohibitively costly. To address this overhead, we propose HIERSAMPLE. Rather than searching one long CDF, we first determine a coarse block containing the random point and then repeat the same search inside that block. Figure 3 illustrates the idea with two hierarchy levels, while our implementation uses three.

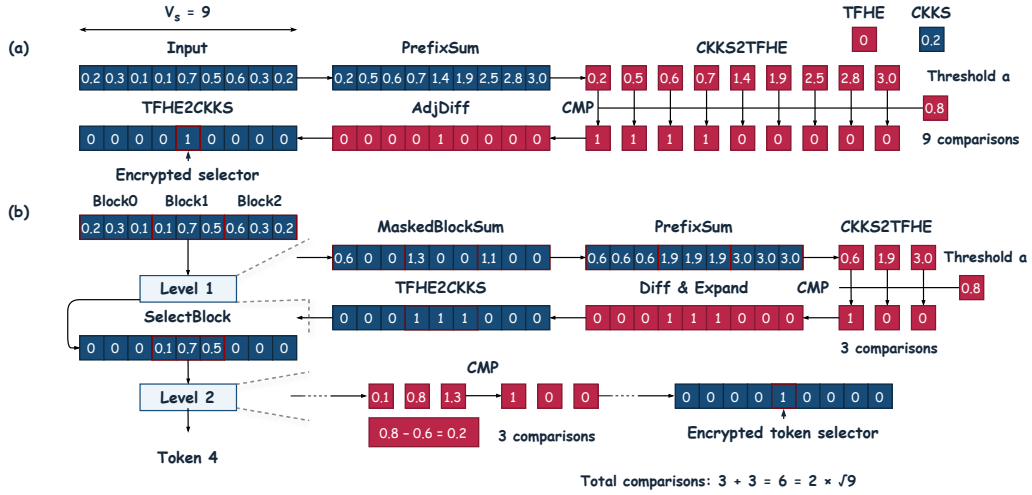

Figure 3: Hierarchical encrypted sampling. (a) A flat inverse-CDF scan requires one comparison per vocabulary entry. (b) HIERSAMPLE first selects a coarse block and then searches within the selected block. In this $V_s = 9$ example, it requires $3 + 3 = 6$ comparisons instead of 9.

More specifically, for an input domain of size V_s, we represent every index using three public digits with radices (m_1, m_2, m_3), where $m_1 m_2 m_3 \geq V_s$. Suppose previous levels have produced encrypted selectors $e_{\ell'}$. At level ℓ, we collect the probability mass of candidate sub-block j as

$$g_\ell[j] = \sum_{i=0}^{V_s-1} w[i]\mathbf{1}\{d_\ell(i) = j\} \prod_{\ell' < \ell} e_{\ell'}[d_{\ell'}(i)], \qquad 0 \leq j < m_\ell. \tag{2}$$

Because all digit maps are public, we form these block masses with plaintext masks and CKKS reductions while keeping the selected branch encrypted. We then build a short CDF, switch the values needed for comparison to TFHE, and obtain an encrypted selector for the chosen block. Figure 3 shows the comparison, selector expansion, and scheme-switching steps. After selecting a block, we need to locate the same random point inside it without drawing a new random number or normalizing its probabilities. We therefore reuse the original threshold throughout the hierarchy. Starting from $a = \omega\|w\|_1$, after selecting block e_ℓ we update

$$a \leftarrow a - \sum_j e_\ell[j] G_\ell[j-1], \qquad G_\ell[-1] = 0. \tag{3}$$

Here, we subtract exactly the probability mass before the selected block, so the updated a represents the same random point measured from the beginning of that block. Repeating this procedure recovers the same token as flat inverse-CDF sampling under exact arithmetic. Figure 3 shows this transformation from the global threshold 0.8 to 0.2 inside the selected block. With this construction, we reduce the number of TFHE comparisons from V_s to $m_1 + m_2 + m_3$. We use $(51, 51, 51)$ for Llama and $(54, 54, 54)$ for Qwen3, requiring 153 and 162 comparisons for native-vocabulary sampling, while the 32K draft vocabulary uses $(32, 32, 32)$ and 96 comparisons. After the final level, we reuse the encrypted selector to retrieve the sampled embedding and record its proposal mass for later verification. Algorithm 2 and Appendix C.1 give the remaining details.

3.3 TARGET FORWARD

Once the draft tree is complete, several candidate states become available before the target forward, providing the parallelism that ordinary autoregressive FHE decoding lacks. With a public tree attention mask, we evaluate these candidates together while keeping each candidate restricted to the common history and its own ancestors. To exploit this parallelism, we place different candidate states in separate CKKS slot regions and evaluate SiLU, Softmax, normalization, and bootstrapping across them together. We keep the reductions for Softmax and normalization within each candidate's slots, while linear projections follow CacheMir's per-token protocol and are distributed across GPUs. The resulting target distributions and model states remain encrypted until verification determines which continuation is retained. Appendix B gives the packing layouts and execution details.

3.4 DIVISION-FREE MSS VERIFICATION

After the target forward, every sampled candidate comes with the draft proposal from which its token was drawn and the corresponding target distribution. We use these two distributions to decide which speculative tokens can be returned without changing the target autoregressive distribution, as illustrated in Figure 4(a). Following SpecInfer's MSS rule (Miao et al., 2024), consider child $v = \mathrm{child}_j(u)$ carrying token x_v. Let $\pi_{u,j}$ denote the proposal used to sample x_v, and let $\mu_{u,j}$ denote the target residual after the preceding siblings have been rejected. We accept the child with probability

$$\min\left\{1, \frac{\mu_{u,j}(x_v)}{\pi_{u,j}(x_v)}\right\}. \tag{4}$$

When the child is rejected, we subtract its proposal from the target residual, keep the positive part, normalize the result, and continue with the next sibling. The first accepted child continues the path, and when every child rejects, the final residual is used to sample the bonus. However, repeatedly normalizing the residual is expensive under FHE because every rejection introduces another encrypted reciprocal, and approximation errors can propagate to later decisions. To avoid these divisions, we keep the target residual as unnormalized weights $r_{u,j}$ with total mass $Z_{u,j} = \|r_{u,j}\|_1$, and the recorded proposal $q_{u,j}$ with total $Q_{u,j} = \|q_{u,j}\|_1$. For a random coin $\eta_{u,j} \in (0,1)$, we accept the child when

$$a_{u,j} = \mathbf{1}\left\{Q_{u,j} r_{u,j}[x_v] > \eta_{u,j} Z_{u,j} q_{u,j}[x_v]\right\}. \tag{5}$$

Indeed, dividing by $Z_{u,j}Q_{u,j}$ recovers exactly the target-to-draft ratio used by MSS. When the child is rejected, we update

$$r_{u,j+1} = \mathrm{ReLU}\left(Q_{u,j} r_{u,j} - Z_{u,j} q_{u,j}\right), \qquad Z_{u,j+1} = \|r_{u,j+1}\|_1. \tag{6}$$

After normalization, this vector gives the same residual as plaintext MSS without evaluating a reciprocal. Figure 4 illustrates the relation between the normalized and unnormalized forms. A second difficulty comes from the control flow. In plaintext, once one sibling is accepted, we follow that child and stop testing its remaining siblings. Under FHE, branching on the encrypted acceptance decision would reveal private information. We therefore evaluate Equation 5 over the fixed public tree and use

$$f_{u,j} = a_{u,j} \prod_{k<j} (1 - a_{u,k}), \qquad s_u = \prod_j (1 - a_{u,j}) \tag{7}$$

to retain the first accepted child and mark the point where all children reject. Starting from the root, we propagate these encrypted decisions through the tree so that one path and one endpoint are selected. As shown in Figure 4(a), the same masks identify the accepted tokens and the endpoint residual, from which we sample the bonus with HIERSAMPLE. Algorithm 3 and Appendix C.3 give the complete fixed-trace procedure.

Theorem 1 (Ideal correctness of VERIFYMSS)**.** *Under exact nonnegative arithmetic, positive masses on every reached state and sampled proposal, and independent uniform random numbers,* VERIFYMSS *produces the same output distribution as autoregressive decoding with the target model.*

Following the local MSS mass identity, Appendix E proves the theorem by induction over the accepted tree path.

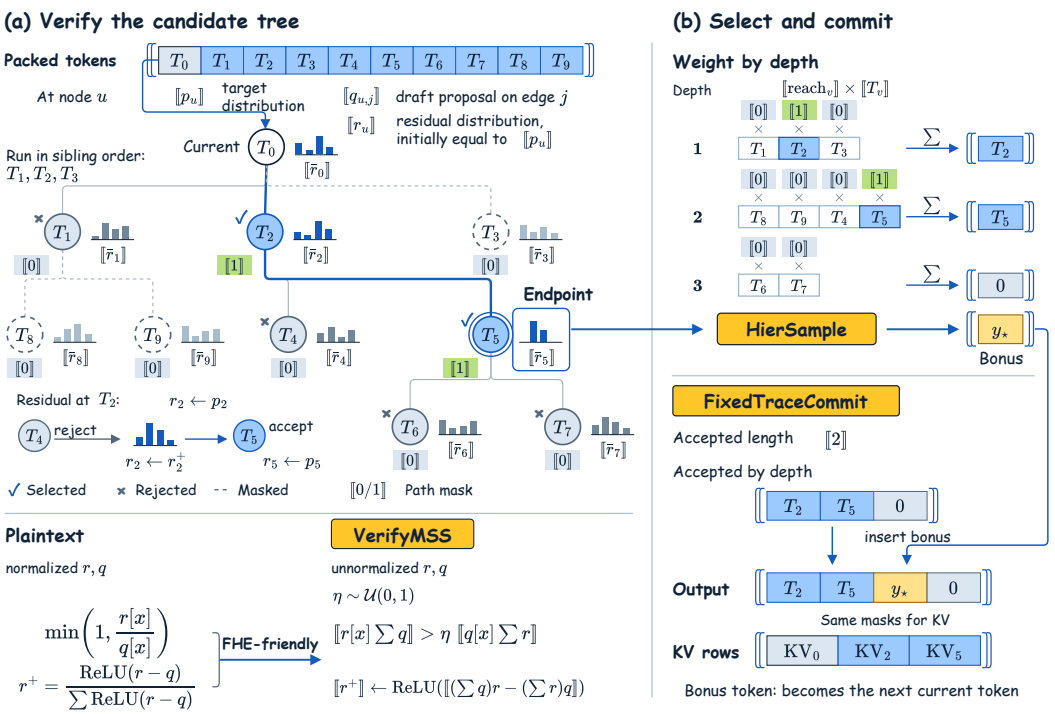

Figure 4: Verification and FIXEDTRACECOMMIT in Taurus. In the example tree, VERIFYMSS accepts tokens T_2 and T_5 while rejecting other candidates, and the endpoint residual is used to sample the bonus token. The same encrypted masks then gather tokens and KV rows by depth without revealing the selected path. The inset relates plaintext MSS normalization to the proposed unnormalized formulation. Token packing and residual bars are schematic.

3.5 FIXED TRACE COMMIT

Finally, after verification finishes, the accepted path remains encrypted while its tokens and KV rows are scattered across the tree. Before the next decoding round, we need to place them into consecutive decoding positions without revealing which nodes were selected. As shown in Figure 4(b), we reuse the encrypted path masks to gather the selected token and KV row at each public depth, write them to consecutive positions, append the bonus after the accepted prefix, and use it as the next decoding round's current token. Because we follow the same accesses and computation for every outcome, neither memory accesses nor execution length reveal the accepted path or its length. Appendices C.3–C.4 give the remaining details.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Implementation. We implement Taurus on eight NVIDIA RTX 4090 GPUs using Cheddar (Choi et al., 2026). Target forward and verification use RNS-CKKS with $N = 2^{16}$, while comparisons use TFHE; Appendix A.1 lists the parameters. During target forward, linear operations invoke CacheMir independently per token across GPUs, while nonlinear functions and bootstrapping process packed candidates.

Models. We use six target models ranging from 1.7B to 8B parameters: Llama-3.1-8B-Instruct, Qwen3-8B, DeepSeek-R1-Distill-Llama-8B, Llama-3.2-3B-Instruct, Qwen3-4B-Instruct-2507, and Qwen3-1.7B (Grattafiori et al., 2024; DeepSeek-AI, 2025; Yang et al., 2025). We denote the target–

draft pairs by P1–P6; each uses a publicly released, pretrained one-layer EAGLE-3-compatible draft (Li et al., 2025a).

Baseline. We compare with CacheMir (Yu et al., 2026), the only open-source baseline for non-interactive FHE autoregressive decoding with an encrypted KV cache. Both systems use the same GPU platform, encrypted backend, and native-vocabulary sampler. CacheMir performs a target forward, sampling, and commit; Taurus adds drafting and verification.

4.2 SAMPLING ABLATION

Hierarchical sampling ablation. We replace HIERSAMPLE with a flat inverse-CDF scan while fixing the rest of the pipeline. Table 1 reports absolute latency and speedup over CacheMir. Testing one boundary per vocabulary entry raises latency to 962.613–1,437.885 seconds per committed token, or 0.07×–0.15× CacheMir throughput. HIERSAMPLE instead reaches 36.357–66.942 seconds and 1.80×–3.02× speedup.

Table 1: Sampling ablation: latency (s/committed token) and speedup over CacheMir.

Sampler	Metric	P1	P2	P3	P4	P5	P6
Flat scan	Latency	962.613	1,054.734	998.424	1,120.600	1,437.885	991.333
	Speedup (×)	0.150	0.139	0.144	0.085	0.096	0.066
HIERSAMPLE	Latency	47.782	53.655	51.192	44.943	66.942	36.357
	Speedup (×)	3.022	2.732	2.809	2.119	2.062	1.800

4.3 VERIFICATION ABLATION

We compare VERIFYMSS with normalized verification, which approximates reciprocals using a degree-27 polynomial. For each model–task pair, we evaluate 192 held-out decoding states, each under 128 shared random seeds (24,576 tests per pair; 442,368 total), while approximation scales are fixed on 64 separate rounds. Across 442,368 tests, VERIFYMSS changes only 0.0228% of accepted paths relative to FP64 MSS, versus 2.7751% for normalized verification. For the decoding-round output, the mismatch rates remain low at 0.0576% for VERIFYMSS, but rise to 3.4763% for normalized verification. This larger error is consistent with 22.34% of its reciprocal inputs falling outside the fitted approximation range.

Table 2: Accepted-path mismatch (ppm) vs. FP64: VERIFYMSS / normalized verification.

Pair	GSM8K	MT-Bench	HumanEval	Pair	GSM8K	MT-Bench	HumanEval
P1	448/29948	0/39266	244/27100	P4	570/50049	448/58838	81/34017
P2	529/42928	0/21932	0/20345	P5	122/9684	0/13713	203/26408
P3	651/37964	407/14852	407/32959	P6	0/7202	0/20101	0/12207

Appendix D.3 reports reciprocal-degree sensitivity and recurrent all-ciphertext agreement with the original FP64 target models.

4.4 CANDIDATE-BUDGET SENSITIVITY

Figure 5 varies n from 4 to 16 for all six model pairs and three tasks. Larger trees offer more continuations but add drafting, sampling, and verification, so we select n by decoding-round latency per expected committed token. Appendix D.1 reports GPU-count comparisons.

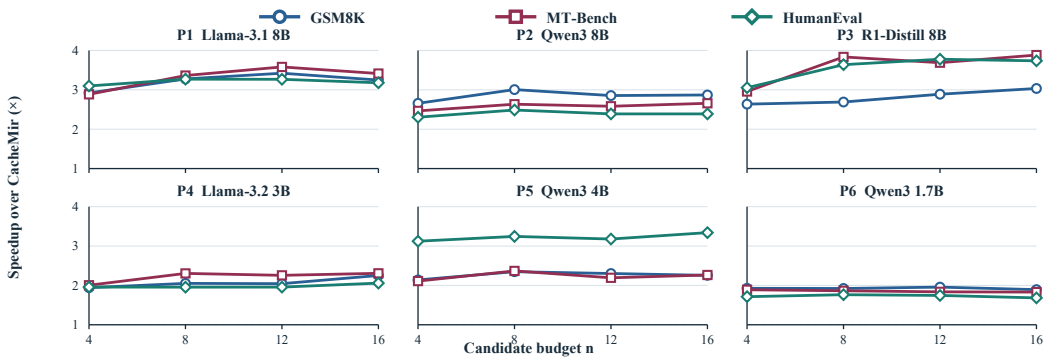

Figure 5: Measured speedup across candidate budgets for six model pairs and three tasks.

4.5 End-to-End Decoder Latency

We measure end-to-end GSM8K decoder latency with $n = 16$ and context length 256, including drafting, sampling, target forward, verification, bonus sampling, and commit.

Taurus produces 2.40–4.01 tokens per target forward and reduces measured latency per committed token, $T_{\text{round}}/\mathbb{E}[1+\tau]$, by 1.80×–3.02× over CacheMir. For P1, 190.604 seconds per decoding round and 3.989 tokens per decoding round yield 47.782 seconds per committed token, versus CacheMir's 144.392 seconds. Figure 6 summarizes all six models; Appendix D.2 gives exact values and stage timings.

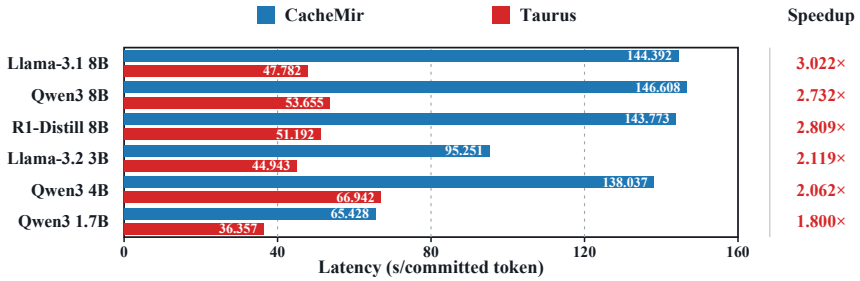

Figure 6: Decoder latency per committed token at $n = 16$; exact values are in Appendix D.2.

5 Conclusion

We present Taurus, a framework that enables speculative decoding for fully homomorphic encrypted LLM inference. Taurus exploits unused ciphertext capacity by packing speculative states and introduces hierarchical encrypted sampling, division-free MSS verification, and FixedTraceCommit to keep the complete decoding loop under FHE. Across six models, Taurus achieves a 1.80×–3.02× speedup in decoder latency per committed token over CacheMir (Yu et al., 2026), while VerifyMSS closely matches FP64 MSS with only 0.0228% accepted-path mismatch across 442,368 paired tests.

AI USE STATEMENT

Generative AI tools assisted with English translation and language editing, manuscript typesetting, figure and table preparation, and the development and testing of experimental code. They also supported literature retrieval, reference formatting, and artifact documentation. We did not use generative AI for algorithm design, experimental design, mathematical proofs, or interpretation of the results, nor to generate datasets or reported measurements. The authors reviewed all AI-assisted work, independently verified the citations, mathematical claims, code, figures, and reported results, and take responsibility for the final submission.

REPRODUCIBILITY STATEMENT

Our code is available at https://anonymous.4open.science/r/Taurus-C7A6/.

REFERENCES

Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic Encryption Standard. Cryptology ePrint Archive, Paper 2019/939, 2019. https://eprint.iacr.org/2019/939.

Song Bian, Zhou Zhang, Haowen Pan, Ran Mao, Zian Zhao, Yier Jin, and Zhenyu Guan. HE3DB: An efficient and elastic encrypted database via arithmetic-and-logic fully homomorphic encryption. In ACM SIGSAC Conference on Computer and Communications Security (CCS), 2023.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In International Conference on Machine Learning (ICML), 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. https://arxiv.org/abs/2302.01318.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yu-Hsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In Advances in Neural Information Processing Systems (NeurIPS), 2024.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT), 2017.

Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In International Conference on Selected Areas in Cryptography (SAC), 2018.

Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. In International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT), 2020.

Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology, 33(1), 2020.

Hyeongmin Choe, Jaehyung Kim, Damien Stehlé, and Elias Suvanto. Leveraging discrete CKKS to bootstrap in high precision. In ACM SIGSAC Conference on Computer and Communications Security (CCS), 2025. doi: 10.1145/3719027.3765080.

Wonseok Choi, Jongmin Kim, and Jung Ho Ahn. Cheddar: A swift fully homomorphic encryption library designed for GPU architectures. In ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pages 35–49, 2026. doi: 10.1145/3760250.3762223.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. arXiv preprint arXiv:2501.12948, 2025. https://arxiv.org/abs/2501.12948.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. The Llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024. https://arxiv.org/abs/2407.21783.

Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in HElib. In Annual International Cryptology Conference (CRYPTO), 2018.

Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In USENIX Security Symposium, 2018.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In International Conference on Machine Learning (ICML), 2023.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test. In Advances in Neural Information Processing Systems (NeurIPS), 2025a.

Zhengyi Li, Yue Guan, Kang Yang, Yu Feng, Ning Liu, Yu Yu, Jingwen Leng, and Minyi Guo. An efficient private GPT never autoregressively decodes. In International Conference on Machine Learning (ICML), volume 267 of Proceedings of Machine Learning Research, pages 34410–34428, 2025b.

Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In IEEE Symposium on Security and Privacy (S&P), pages 1057–1073, 2021. doi: 10.1109/SP40001.2021.00043.

Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and WenGuang Chen. BumbleBee: Secure two-party inference framework for large transformers. In Network and Distributed System Security Symposium (NDSS), 2025. doi: 10.14722/ndss.2025.230057.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating generative large language model serving with tree-based speculative inference and verification. In ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2024.

Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. THOR: Secure transformer inference with homomorphic encryption. In ACM SIGSAC Conference on Computer and Communications Security (CCS), 2025. doi: 10.1145/3719027.3765150.

Dongjin Park, Eunsang Lee, and Joon-Woo Lee. Powerformer: Efficient and high-accuracy privacy-preserving language model with homomorphic encryption. In Annual Meeting of the Association for Computational Linguistics (ACL), pages 11090–11111, 2025. doi: 10.18653/v1/2025.acl-long.543.

Jaiyoung Park, Sejin Park, Jai Hyun Park, Jung Ho Ahn, Jung Hee Cheon, Guillaume Hanrot, Jung Woo Kim, Minje Park, and Damien Stehlé. Scaling up FHE-based privacy-preserving ML: Higher throughput, longer inputs for LLama-3-8B. arXiv preprint arXiv:2601.18511, 2026. https://arxiv.org/abs/2601.18511.

An Yang et al. Qwen3 Technical Report. arXiv preprint arXiv:2505.09388, 2025. https://arxiv.org/abs/2505.09388.

Ye Yu, Yifan Zhou, Yi Chen, Pedro Soto, Wenjie Xiong, and Meng Li. CacheMir: Fully homomorphic encrypted inference of generative large language model with KV cache. arXiv preprint arXiv:2602.11470, 2026. https://arxiv.org/abs/2602.11470.

Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. Secure transformer inference made non-interactive. In Network and Distributed System Security Symposium (NDSS), 2025. doi: 10.14722/ndss.2025.230868.

A Cryptographic Parameters and Notation

A.1 Cryptographic Parameters

Table 3: Cryptographic parameters.

Component	Scheme	Dimension	Modulus basis	Total modulus
Model Forward	CKKS	$N = 2^{16}$	48 primes, 20–31 bits	$\log_2 Q = 1370$
Comparison	TFHE	$N_{\mathrm{TFHE}} = 2^{10}$	1 torus modulus, 32 bits	$\log_2 q = 32$

In our CKKS configuration, the auxiliary basis P contains twelve 31-bit primes ($\log_2 P = 371.93$), and the default scale is $\Delta = 2^{30}$. The CKKS parameter sets satisfy at least 128-bit classical security under the Homomorphic Encryption Standard (Albrecht et al., 2019), while TFHE uses the Cheddar library's default 128-bit preset.

A.2 Notation and Protocol Functions

Table 4: Notation and protocol functions. Tree indices and layouts are public; token values and protocol decisions remain encrypted.

Symbol or function	Meaning
M_p, M_q	Target and draft models.
V, V_d, V_s	Native vocabulary size, compact draft vocabulary size, and domain size of a sampler call.
$\phi(i)$	Fixed map from draft entry i to its native token; identity for native sampling.
$\mathcal{T}, n, D$	Ordered tree, number of non-root candidates, and maximum depth; node 0 is the root.
u, v, j, t, ℓ, r	Parent, node, sibling rank, path depth, sampling level, and decoding round.
i, k, x	Sampling or vector indices and native token value.
$\mathrm{children}(u)$, $\mathrm{child}_j(u)$	Ordered children and the j-th child of u.
$\mathrm{depth}(v)$	Public depth of node v.
$x_v, y_t, y_\star$	Node token, accepted token, and bonus token.
E_i, E_x, M_x	Input-domain and native one-hot selectors; $M_x[i] = E_x[\phi(i)]$.
$\mathrm{Tok}(x)$, Tok_v	Encrypted native selector and short token digits; $\mathrm{Tok}_v = \mathrm{Tok}(x_v)$.
p_v, Z_v	Target next-token weights and total $Z_v = \|p_v\|_1$.
$q_{u,j}$	Native proposal masses recorded when child j of u is sampled.
$\pi_{u,j}, \mu_{u,j}$	Normalized draft proposal and target residual used by MSS.
$w, m_\ell, d_\ell(i)$	Input vector, level radix, and digit map; w may be signed in ReLU.
g_ℓ, G_ℓ, a	Block masses, cumulative block masses, and remaining sampling threshold.
$\widetilde{b}_\ell, \widetilde{e}_\ell, e_\ell$	TFHE boundary-comparison bits, their TFHE adjacent-difference selector, and its CKKS representation.
$\omega, \omega_\star, \eta_{u,j}$	Random numbers for draft sampling, bonus sampling, and MSS acceptance.
$r_{u,j}, Z_{u,j}, \bar{r}_u$	Target residual weights, their total, and the residual after all children of u reject.

Table 4 (continued)

Symbol or function	Meaning
$Q_{u,j}$	Total mass of recorded proposal $q_{u,j}$.
$a_{u,j}, f_{u,j}, s_u$	MSS accept bit, first-accepted-child bit, and all-rejected bit.
reach_u, z_u, z	Reached-node bit, endpoint bit, and endpoint one-hot vector.
τ	Number of accepted candidate tokens in one decoding round.
Σ_r	Target and draft caches, H_r, P_r, and current token $\mathrm{Tok}_{0,r}$.
$R_v^h = (K_v^h, V_v^h)$	KV row for node v in model $h \in \{t, d\}$.
H_r, H_v	Saved target feature before the current token and feature produced at node v.
P_r	Encrypted position one-hot.
m_t, λ_t, C_t	Accepted-prefix indicator, accepted-length one-hot, and decoding-round output token.
$N, N_{\mathrm{TFHE}}, S, C, d_{\mathrm{ffn}}$	CKKS degree, TFHE LWE dimension, slots $S = N/2$, GPU count, and FFN intermediate width.
$\rho, w_\rho, k_\rho, W_\rho$	Bootstrap site, slots per token, ciphertext count, and nominal refresh batches.
$T_{\mathrm{round}}, T_{\mathrm{AR}}, \mathbb{E}[1+\tau]$	Decoding-round latency, baseline autoregressive-step latency, and expected tokens per decoding round.
$H_{a,b}, f_7, g_7$	Half-sign approximation and its composition polynomials.
$L, V_r[j], u_j, u_{r,t}$	Generic hierarchy depth, cache validity, public rotary coefficients, and selected coefficients.
TFHECompare, CKKS2TFHE, TFHE2CKKS	Strict fixed-point comparison and scheme-switching operations.
$\mathrm{Rot}(P, -t)$	Fixed rotation advancing an encrypted position by t.
InvCDF, HierSample	Sampling rule in Equation 1 and its hierarchical encrypted implementation.
DRAFTTREE, PACKEDTARGET	Tree drafting and target forward over the root and candidates.
VERIFYMSS, FIXEDTRACECOMMIT	Fixed-trace encrypted MSS verification and commit.
$Z_w, \ell', \widetilde{L}_\ell, L_\ell, L_{1:\ell-1}$	Sampler input total $Z_w = \|w\|_1$, earlier level, TFHE and CKKS expanded digit masks, and preceding masks.
$r_\star$	Token residual selected for bonus sampling.
$\mathrm{ReduceSum}(w)$, $\mathrm{GatherMass}(w, E)$	$\sum_i w[i]$ and $\sum_i w[i]E[i]$.
$\mathrm{ReLU}(w)$	Element-wise rectification, implemented by the CKKS polynomial in Appendix C.1.
MASKEDBLOCKSUM, PREFIXSUM	Block sums in Equation 2 and cumulative block sums.
$\mathrm{AdjacentDifference}_{\mathrm{TFHE}}(\widetilde{b})$	$\widetilde{e}[j] = \widetilde{b}[j-1] - \widetilde{b}[j]$ in TFHE, with $\widetilde{b}[-1] = 1$.
EXPANDSELECTOR	Expands digit selectors to domain masks and maps them to native selectors.
SELECTANDUPDATECARRY	Subtracts $\sum_j e_\ell[j] G_\ell[j-1]$, with $G_\ell[-1] = 0$.

B PACKING AND TARGET FORWARD

Motivating breakdown. We first examine the computation that speculation can share across candidate states. In CacheMir's reported CPU breakdown for one Llama-3-8B decoder block, Softmax, SiLU, the two Add & Norm modules, and bootstrapping take 410.64 of the 424.72 seconds, account-

ing for 96.7% of decoder-block latency (Yu et al., 2026, Table 5). The Add & Norm entries include residual additions, and the bootstrapping entry accounts for ciphertext refreshes. These operations motivate packing distinct candidate states for nonlinear evaluation while retaining CacheMir's per-token linear projections.

Packing capacity. The packing layout accommodates the intermediate tensors of each operation. At bootstrap site ρ, let w_ρ denote the slots occupied by one token state. Packing the root and n candidates requires

$$k_\rho(n) = \left\lceil \frac{(n+1)w_\rho}{S} \right\rceil, \qquad W_\rho(n) = \left\lceil \frac{k_\rho(n)}{C} \right\rceil. \tag{8}$$

Here $k_\rho(n)$ is the ciphertext count, and $W_\rho(n)$ is the number of parallel refresh batches when each of the C GPUs processes one ciphertext per batch. For Llama-3.1-8B with 16 candidates, hidden states occupy eight ciphertexts. Equation 8 therefore gives the capacity available at each site, allowing us to select a tree whose nonlinear operations fit the available parallelism.

For each linear projection, we invoke CacheMir's ciphertext-plaintext vector-matrix multiplication protocol separately for the root and every candidate. Independent invocations at the same projection are scheduled across GPUs. This applies to attention projections, FFN projections, and the output head. Fixed layout conversions connect these token-specific projection layouts to the packed nonlinear layouts. Attention also evaluates each query against its cached and ancestor keys and combines the corresponding values under the tree mask.

However, assigning ciphertext storage to complete tokens would leave useful capacity idle. We therefore pack compatible intermediate fragments together before scheduling the corresponding ciphertext operations. For the FFN intermediates of P1, the 17 states occupy

$$17 \cdot d_{\text{ffn}} = 17 \cdot 14{,}336 = 243{,}712$$

slots. Packing across state boundaries requires $\lceil 243{,}712/32{,}768 \rceil = 8$ ciphertexts. These fit into one parallel batch on eight GPUs; this count covers one intermediate at a single bootstrap site.

Tree selection. Packing capacity determines the available concurrency, whereas the assigned projections, attention products, nonlinear operations, and layout conversions determine the work performed during target forward. We evaluate both single-token and multi-token target forwards at their respective workload sizes. Table 7 shows how the longer target forward can still reduce latency per committed token.

To select the static tree, we minimize decoding-round latency per expected committed token,

$$\min_{\mathcal{T}} \frac{T_{\text{round}}(\mathcal{T})}{\mathbb{E}[1+\tau(\mathcal{T})]} \quad \text{subject to } k_\rho(n) \le C \text{ at every bootstrap site } \rho. \tag{9}$$

The objective covers target forward together with drafting, sampling, verification, and commit. If T_{AR} denotes CacheMir's autoregressive-step latency under the same configuration, speculation improves decoder latency per committed token when

$$T_{\text{round}}(\mathcal{T}) < \mathbb{E}[1+\tau(\mathcal{T})]\, T_{\text{AR}}.$$

A larger tree can therefore improve decoding even when it lengthens target forward and the decoding round, provided that the additional accepted tokens compensate for this increase. We estimate acceptance on development prompts and fix the tree before evaluation.

C Encrypted Protocol Operations

Algorithms 1–3 collect the three fixed-schedule protocols used in one Taurus decoding round. Algorithm 1 first shows how drafting, packed target forward, verification, and commit connect without revealing any intermediate decision.

Algorithm 1 One server-side Taurus round.

Input: Encrypted state Σ_r; ordered tree $\mathcal{T}$; target and draft weights; independent server-held random numbers.
Output: Updated encrypted state Σ_{r+1}.

$(\{\mathrm{Tok}_v\}_{v\neq 0}, \{q_{u,j}\}, \{R_v^d\}_{v\in\mathcal{T}}) \leftarrow \mathrm{DraftTree}(\Sigma_r, \mathcal{T})$
$\{p_v, Z_v, R_v^t, H_v\}_{v\in\mathcal{T}} \leftarrow \mathrm{PackedTarget}(\Sigma_r, \{\mathrm{Tok}_v\}, \mathcal{T})$
$(z, \tau, \{m_t, \mathrm{Tok}(y_t)\}, \mathrm{Tok}(y_\star)) \leftarrow \mathrm{VerifyMSS}(\mathcal{T}, \{p_v, Z_v\}, \{q_{u,j}\}, \{\mathrm{Tok}_v\}, \{\eta_{u,j}\}, \omega_\star)$
$\Sigma_{r+1} \leftarrow \mathrm{FixedTraceCommit}(\Sigma_r, \{z_v, m_t, \mathrm{Tok}(y_t)\}, \mathrm{Tok}(y_\star), \{R_v^t, R_v^d, H_v\})$
return Σ_{r+1}

C.1 Sampling Layout and Packed ReLU

The sampling hierarchy divides a large cumulative distribution into several shorter ones. Each additional level reduces the number of boundaries at the expense of another sequential scheme-switching stage and another selector update. A balanced L-level hierarchy uses approximately $LV_s^{1/L}$ boundary comparisons, so we use three levels for both draft and target sampling. Native target sampling uses 153 boundary comparisons for Llama and 162 for Qwen3. Every level aggregates all vocabulary entries in CKKS and sends only its short cumulative list to TFHE. We compare the threshold with this list, take the adjacent difference of the comparison bits, and replicate the resulting selector according to the public digit map in TFHE; the selector and expanded mask are then switched back to CKKS.

Algorithm 2 gives the complete three-level sampler. The returned native selector is reused to gather the embedding, the target or proposal mass at the sampled token, and the corresponding model state.

Algorithm 2 HierSample, fixed-layout inverse-CDF sampling.

Input: CKKS-encrypted masses $w \geq 0$ with positive total; server-held coin $\omega \in (0, 1]$; fixed radices $m_{1:3}$, digit maps $d_{1:3}$, and map ϕ to native tokens.
Output: Encrypted digit selectors $e_{1:3}$; input-domain one-hot E_i for Equation 1; native-token selector E_x; and domain mask M_x.

$Z_w \leftarrow \mathrm{ReduceSum}(w)$; $\quad a \leftarrow \omega Z_w$
for $\ell = 1, 2, 3$ **do**
  $g_\ell \leftarrow \mathrm{MaskedBlockSum}(w, d_\ell, L_{1:\ell-1})$
  $G_\ell \leftarrow \mathrm{PrefixSum}(g_\ell)$
  $(\widetilde{a}, \widetilde{G}_\ell) \leftarrow \mathrm{CKKS2TFHE}(a, G_\ell)$
  $\widetilde{b}_\ell \leftarrow \mathrm{TFHECompare}(\widetilde{a}, \widetilde{G}_\ell)$ ▷ $\widetilde{b}_\ell[j]$ encrypts $\mathbf{1}\{a > G_\ell[j]\}$
  $\widetilde{e}_\ell \leftarrow \mathrm{AdjacentDifference}_{\mathrm{TFHE}}(\widetilde{b}_\ell)$ ▷ TFHE one-hot for the first crossed boundary
  $\widetilde{L}_\ell \leftarrow \mathrm{ExpandSelector}(\widetilde{e}_\ell, d_\ell)$ ▷ public replication of TFHE selector bits
  $(e_\ell, L_\ell) \leftarrow \mathrm{TFHE2CKKS}(\widetilde{e}_\ell, \widetilde{L}_\ell)$ ▷ CKKS selector and expanded mask
  $a \leftarrow \mathrm{SelectAndUpdateCarry}(a, G_\ell, e_\ell)$
end for
$(E_i, E_x, M_x) \leftarrow \mathrm{ExpandSelector}(e_{1:3}, d_{1:3}, \phi)$
return $(e_{1:3}, E_i, E_x, M_x)$

For siblings sampled without replacement, we additionally mask the selected native token from the remaining proposal.

For a rejected child, Equation 6 applies ReLU coordinatewise to $w = Q_{u,j}r_{u,j} - Z_{u,j}q_{u,j}$. We evaluate this operation as

$$\mathrm{ReLU}_{\mathrm{CKKS}}(w) = w \odot H_{20,3}(w/2),$$

where the half-sign polynomial $H_{a,b}$ composes two degree-seven polynomials,

$$\begin{aligned} f_7(x) &= (35x - 35x^3 + 21x^5 - 5x^7)/16, \\ g_7(x) &= (4589x - 16577x^3 + 25614x^5 - 12860x^7)/1024, \\ H_{a,b}(x) &= \big(1 + f_7^{\circ b}(g_7^{\circ a}(x))\big)/2. \end{aligned}$$

One packed polynomial evaluation processes every coordinate of a residual ciphertext. A compact proposal is zero outside the image of ϕ; on this complement, the rejection update reduces to $Q_{u,j} r_{u,j}$, so the full native residual remains available for bonus sampling.

C.2 Unnormalized MSS Residuals

For parent u, $r_{u,j}/Z_{u,j}$ is the target residual conditioned on rejection of the first j siblings, while $q_{u,j}/Q_{u,j}$ is the proposal from which child j was sampled. Multiplying either vector and its total by a common positive scale leaves Equation 5 unchanged. Likewise, Equation 6 equals $Z_{u,j}Q_{u,j}$ times the positive part of the difference between these two normalized distributions. Its normalization is therefore represented by the next pair $(r_{u,j+1}, Z_{u,j+1})$ without evaluating a reciprocal.

During encrypted execution, a fixed sequence of threshold tests selects a common power-of-two representation scale before the next comparison. The selected factor is applied jointly to a residual and its total, and every threshold test and multiplication follows the same public schedule. Algorithm 3 presents the ideal mass recurrence and omits this representation bookkeeping.

C.3 Fixed-Trace MSS Routing and FixedTraceCommit

Algorithm 3 evaluates the local MSS recurrence for every parent, including parents that are not ultimately reached. The encrypted routing masks then retain only the first accepted child along the selected path.

Algorithm 3 VerifyMSS, fixed-trace encrypted multi-step speculative sampling.

Input: Ordered tree $\mathcal{T}$; encrypted target masses (p_u, Z_u); recorded proposals $q_{u,j}$; encrypted node tokens Tok_v containing E_{x_v}; acceptance coins $\eta_{u,j} \in (0,1)$; bonus coin $\omega_\star$.
Output: Endpoint selector z; accepted length τ; accepted tokens $\mathrm{Tok}(y_t)$; bonus token $\mathrm{Tok}(y_\star)$.

for all $u \in \mathrm{Parents}(\mathcal{T})$ **in parallel do**
  $r_{u,0} \leftarrow p_u$; $Z_{u,0} \leftarrow Z_u$ ▷ target residual and its mass
  for $j = 0, \dots, |\mathrm{children}(u)| - 1$ **do** ▷ public drafting order
    $v \leftarrow \mathrm{child}_j(u)$; $Q_{u,j} \leftarrow \mathrm{ReduceSum}(q_{u,j})$
    $r_x \leftarrow \mathrm{GatherMass}(r_{u,j}, E_{x_v})$; $q_x \leftarrow \mathrm{GatherMass}(q_{u,j}, E_{x_v})$ ▷ candidate-token masses
    $a_{u,j} \leftarrow \mathbf{1}\{Q_{u,j} r_x > \eta_{u,j} Z_{u,j} q_x\}$ ▷ one encrypted MSS decision
    $r_{u,j+1} \leftarrow \mathrm{ReLU}(Q_{u,j} r_{u,j} - Z_{u,j} q_{u,j})$ ▷ residual if this child rejects
    $Z_{u,j+1} \leftarrow \mathrm{ReduceSum}(r_{u,j+1})$
  end for
  $\bar{r}_u \leftarrow r_{u,|\mathrm{children}(u)|}$ ▷ residual after all siblings reject
end for
for all $u \in \mathrm{Parents}(\mathcal{T})$ **do**
  $f_{u,j} \leftarrow a_{u,j} \prod_{k<j}(1 - a_{u,k})$ for every j; $s_u \leftarrow \prod_j (1 - a_{u,j})$ ▷ first accept or stop
end for
for all $v \in \mathrm{Leaves}(\mathcal{T})$ **do**
  $\bar{r}_v \leftarrow p_v$; $s_v \leftarrow 1$ ▷ a reached leaf is an endpoint
end for
$\mathrm{reach}_0 \leftarrow 1$; $\mathrm{reach}_v \leftarrow \mathrm{reach}_u\, f_{u,j}$ for $v = \mathrm{child}_j(u)$ ▷ top-down encrypted path
$z_u \leftarrow \mathrm{reach}_u\, s_u$ for all $u \in \mathcal{T}$ ▷ unique endpoint
for $t = 1, \dots, D$ **do**
  $m_t \leftarrow \sum_{\mathrm{depth}(v)=t} \mathrm{reach}_v$; $\mathrm{Tok}(y_t) \leftarrow \sum_{\mathrm{depth}(v)=t} \mathrm{reach}_v\, \mathrm{Tok}_v$ ▷ accepted token at depth t
end for
$\tau \leftarrow \sum_{t=1}^{D} m_t$; $r_\star \leftarrow \sum_{u \in \mathcal{T}} z_u \bar{r}_u$ ▷ bonus residual at the endpoint
$\mathrm{Tok}(y_\star) \leftarrow \mathrm{HierSample}(r_\star, \omega_\star)$ ▷ target bonus
return $(z, \tau, \{m_t, \mathrm{Tok}(y_t)\}, \mathrm{Tok}(y_\star))$

For each parent, the first-accept bits and all-reject bit satisfy $s_u + \sum_j f_{u,j} = 1$. We set $\mathrm{reach}_0 = 1$ at the root and compute

$$\mathrm{reach}_v = \mathrm{reach}_u \, f_{u,j}, \qquad z_u = \mathrm{reach}_u \, s_u, \quad v = \mathrm{child}_j(u).$$

For a leaf, $s_u = 1$ and $\bar{r}_u = p_u$. Thus, a reached internal node either sends reachability to exactly one child or becomes the endpoint, and exactly one encrypted endpoint bit z_u is one. The selected residual $r_\star = \sum_u z_u \bar{r}_u$ is passed directly to HIERSAMPLE for the bonus.

The reach bits also recover the accepted path without a private memory access. For depth t, we compute

$$m_t = \sum_{v:\mathrm{depth}(v)=t} \mathrm{reach}_v, \qquad \mathrm{Tok}(y_t) = \sum_{v:\mathrm{depth}(v)=t} \mathrm{reach}_v \, \mathrm{Tok}_v \, . \tag{10}$$

The indicator m_t equals one precisely when the accepted path reaches depth t. Setting $m_0 = 1$, $m_{D+1} = 0$, and $\lambda_t = m_t - m_{t+1}$ gives a one-hot encoding of the accepted length. We use this encoding to place the bonus immediately after the accepted tokens,

$$C_t = m_t \, \mathrm{Tok}(y_t) + \lambda_{t-1} \, \mathrm{Tok}(y_\star) \quad (1 \le t \le D), \qquad C_{D+1} = \lambda_D \, \mathrm{Tok}(y_\star). \tag{11}$$

The first $\tau + 1$ entries are $y_1, \ldots, y_\tau, y_\star$, followed by zeros. The same reach bits gather the target and draft KV rows at each accepted depth. We append the root rows followed by these path rows, with each cache retaining its model-specific layout.

FIXEDTRACECOMMIT writes the selected tokens and rows by sweeping the allocated buffers in a fixed order. The encrypted position one-hot P_r selects their placement arithmetically. The accepted-length selector advances this position through fixed rotations,

$$P_{r+1} = \sum_{j=0}^{D} \lambda_j \, \mathrm{Rot}(P_r, -(j+1)).$$

The bonus becomes the next current token, and the target feature at the selected endpoint supplies H_{r+1}. Both caches consequently contain the history immediately before that token, as required by Section 3.1. When $\tau = 0$, the same procedure appends the root rows and advances to the bonus.

C.4 PRIVATE POSITIONS

The next decoding round uses the encrypted one-hot P_r to represent the current token's absolute position. Its suffix sums determine which cache rows precede that token,

$$V_r[j] = \sum_{k>j} P_r[k].$$

The server scans every allocated cache row and applies this encrypted validity mask to its attention weight before Softmax normalization. Temporary tree rows use the public ancestor mask. Thus, each query attends to the valid shared history and its own ancestors.

Positional encoding uses the same private position. Let u_j contain the public sine and cosine coefficients for position j. A candidate at public depth t uses

$$u_{r,t} = \sum_j P_r[j] u_{j+t}.$$

Both the positional coefficients and cache validity are selected through arithmetic on the encrypted position. Public table lengths and buffer capacities accommodate the configured sequence length and maximum candidate depth, while the target and draft models retain their own physical cache layouts.

C.5 Privacy Scope

We consider data and control-flow privacy against a semi-honest server. For fixed public model dimensions, tree, and capacities, the server observes ciphertexts and the same sequence of operations and memory accesses for every private outcome. CKKS and TFHE confidentiality protect the values, while encrypted selectors hide the accepted path and its length. The client receives generated tokens, with target logits, draft distributions, and intermediate states retained on the server. In our setting, the client sends a stop/continue signal to the server after each decoding round, so the number of decoding rounds is not fixed, and our security model permits this disclosure.

D Supporting Results

D.1 Packing and Tree Selection

We compare decoding performance on systems with one, two, four, and eight GPUs, using the measured eight-GPU configurations as the reference (Figure 7). More GPUs allow additional per-token projection calls and packed ciphertext operations to run concurrently. Each GPU configuration uses a fixed candidate budget.

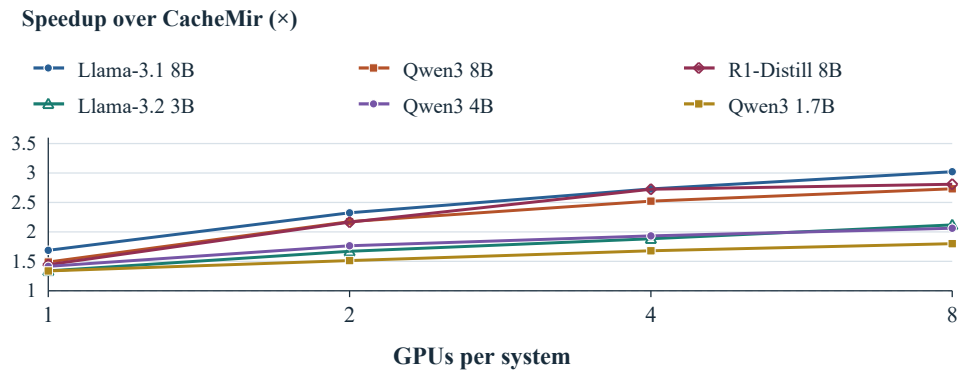

Figure 7: Speedup over CacheMir on GSM8K with one, two, four, and eight GPUs per system.

Table 5 compares four replicated and four distinct Qwen3-1.7B state blocks under the same slot budget. Both layouts occupy 8,192 hidden-state slots and 24,576 FFN slots, requiring one ciphertext at each stage. The replicated layout carries one distinct token state, whereas the speculative layout carries four. Packed nonlinear evaluation processes these distinct states together, while linear operations retain the per-token linear protocol.

The tree structure determines how much useful output this capacity produces. Table 6 compares the selected Taurus tree with an encrypted chain and a fixed binary tree on P6 latency. The selected depth-4 tree produces 2.404 tokens per decoding round, compared with 1.970 for the chain and 2.190 for the binary tree. Since the structures also introduce different amounts of drafting and verification, we compare their latencies per committed token to assess the overall benefit.

Table 5: Packing capacity for Qwen3-1.7B with $S = 32{,}768$, hidden width 2,048, and FFN width 6,144. Entries are slot and ciphertext counts.

Layout	Useful states	Hidden slots	FFN slots	Hidden ciphertexts	FFN ciphertexts
Four copies of one state	1	8,192	24,576	1	1
Four distinct states	4	8,192	24,576	1	1

Table 6: P6 latency comparison on different tree structures. The CacheMir baseline is 65.428 seconds per committed token.

Method	n	D	Tokens per decoding round	Latency (s)	Speedup ($\times$)
Encrypted chain	16	16	1.970	49.367	1.325
Fixed binary tree	16	8	2.190	42.548	1.538
Taurus tree	16	4	2.404	36.357	1.800

D.2 End-to-End and Stage Timings

Table 7 gives the exact end-to-end measurements underlying Figure 6. Table 8 then presents stage timings in seconds per decoding round, before amortization by the mean token yield. The end-to-end values and stage timings come from the same complete encrypted decoding-round executions. Target forward is the largest stage in all six configurations. Drafting, sampling, verification, and commit explain why the final speedup is smaller than the number of tokens produced per decoding round. Drafting includes draft sampling; bonus sampling and commit are reported separately. The final row gives the measured decoding-round latency.

Table 7: Measured end-to-end encrypted decoder latency per committed token on GSM8K, with $n = 16$ and a context length of 256. Each Taurus decoding round includes drafting, sampling, one target forward, verification, and commit.

Model	Tree depth	Tokens per decoding round	CacheMir (s/committed token)	Taurus (s/committed token)	Speedup ($\times$)
P1	7	3.989	144.392	47.782	3.022
P2	8	3.601	146.608	53.655	2.732
P3	10	4.011	143.773	51.192	2.809
P4	8	3.095	95.251	44.943	2.119
P5	7	2.829	138.037	66.942	2.062
P6	4	2.404	65.428	36.357	1.800

Table 8: Taurus stage timings in seconds per decoding round.

Stage	P1	P2	P3	P4	P5	P6
Target forward	151.108	149.626	151.286	97.244	146.751	64.673
Drafting	33.849	37.580	48.421	36.340	36.825	17.100
Verification	1.524	1.610	1.523	1.547	1.499	1.508
Bonus sampling	3.299	3.445	3.289	3.296	3.407	3.456
Commit	0.824	0.929	0.807	0.686	0.879	0.679
Decoding round	190.604	193.189	205.325	139.114	189.360	87.416

D.3 Additional Verification Results

Recurrent all-ciphertext agreement. For each of P1 and P6, we execute 10 consecutive complete encrypted decoding rounds on 16 distinct prompts, giving 160 closed-loop decoding rounds per model pair and 320 decoding rounds in total. Every decoding round consumes the encrypted token and KV-cache state produced by its predecessor. Under the same cryptographic configuration used for latency measurement, we compare every decrypted output token with the original FP64 target model using matched prompts, trees, and sampling randomness. We observe zero output-token mismatches for both model pairs (Table 9). These recurrent tests cover complete model execution, encrypted sampling and verification, token selection, and commit.

Table 9: Recurrent all-ciphertext output agreement with the original FP64 target models. Every decoding round consumes the encrypted token and KV-cache state produced by its predecessor.

	P1	P6	Total
Distinct prompts	16	16	32
Consecutive decoding rounds per prompt	10	10	–
All-ciphertext decoding rounds	160	160	320
Output-token mismatches	0	0	0

Reciprocal-degree sensitivity. We compare normalized recurrences with Chebyshev interpolants of $1/x$ under the same recorded masses, random numbers, and $H_{20,3}$ CKKS ReLU approximation. Exact inverse-CDF bonus sampling isolates the repeated reciprocal approximation. Table 10 shows that these alternatives change 2.547% to 13.060% of the combined verification and bonus outputs relative to FP64, motivating the use of unnormalized masses.

Table 10: Normalized verification ablation with $H_{20,3}$ against the FP64 reference.

Verification method	Reciprocal degree	Approximation interval	Output mismatch (%)
Normalized recurrence	27	$[9.1 \times 10^{-3}, 1]$	2.547
Normalized recurrence	59	$[2.2 \times 10^{-3}, 1]$	13.060
Normalized recurrence	119	$[5.3 \times 10^{-4}, 1]$	7.324

D.4 CKKS-Based Sampling

We also examine whether deeper CKKS sign approximations can replace TFHE comparisons in sampling. The half-sign polynomial $H_{a,b}$ and its composition counts are defined in Appendix C.1. Table 11 reports token mismatches, invalid selectors, and decoding-round output mismatches for P6 with exact MSS decisions, isolating the effect of the sampling comparisons. Increasing the composition counts reduces some errors, but token mismatches and invalid selectors persist in the evaluated configurations.

Table 12 presents selected ordinary and near-boundary cases across all six model pairs. These examples show how approximation error around a cumulative boundary can change a discrete sampling decision. Together with the hierarchical comparison in Section 4.2, the results support retaining CKKS for packed vector arithmetic and using TFHE for the comparisons.

Table 11: Native-CKKS sampling on P6. Entries give mismatches or invalid selectors; MSS decisions remain exact.

CKKS sign	Token mismatch	Invalid selector	Decoding-round output mismatch
$H_{1,1}$	96/96	96/96	48/48
$H_{2,2}$	20/96	94/96	25/48
$H_{5,2}$	16/96	21/96	17/48
$H_{20,3}$	13/96	11/96	23/48

Table 12: Native-CKKS $H_{20,3}$ sampling checks across model pairs. Ordinary and near-boundary cases are shown separately.

Pair	Ordinary: token mismatch	Ordinary: invalid	Near-boundary: token mismatch	Near-boundary: invalid
P1	1/8	1/8	1/2	0/2
P2	1/8	1/8	1/2	1/2
P3	1/8	1/8	2/2	1/2
P4	4/8	4/8	2/2	1/2
P5	3/8	2/8	2/2	2/2
P6	2/8	2/8	2/2	2/2

E IDEAL CORRECTNESS OF VERIFYMSS

Proof of Theorem 1. We assume exact nonnegative arithmetic. Every live candidate is sampled from its recorded proposal, siblings retain their drafting order, and the drafting, acceptance, and bonus coins are independent. Target residuals and live proposals have positive total mass; a sampled token consequently has positive proposal mass.

Equivalence of the unnormalized recurrence. Before testing child $v = \mathrm{child}_j(u)$, define the normalized target residual and proposal as

$$\mu_{u,j} = \frac{r_{u,j}}{Z_{u,j}}, \qquad \pi_{u,j} = \frac{q_{u,j}}{Q_{u,j}}.$$

For the sampled token x_v, Equation 5 is equivalent to

$$\eta_{u,j} < \frac{\mu_{u,j}(x_v)}{\pi_{u,j}(x_v)}.$$

Because $\eta_{u,j} \in (0,1)$, this accepts x_v with probability $\min\{1, \mu_{u,j}(x_v)/\pi_{u,j}(x_v)\}$, exactly as in MSS. If the candidate is rejected, Equation 6 gives

$$r_{u,j+1} = Z_{u,j} Q_{u,j} [\mu_{u,j} - \pi_{u,j}]_+.$$

After division by its total, this is the normalized MSS residual. If the positive part has zero total, the corresponding rejection event has probability zero and the later branch is unreachable. Thus, for the same drafted candidates and random coins, the unnormalized circuit implements the normalized MSS tests without evaluating a reciprocal.

Local MSS identity. Consider one test with current target residual μ, proposal π, and candidate $X \sim \pi$. For any token x, the probability mass that accepts and outputs x is

$$\pi(x) \min\left\{1, \frac{\mu(x)}{\pi(x)}\right\} = \min\{\pi(x), \mu(x)\}.$$

The rejection probability is

$$\delta = \sum_x [\pi(x) - \mu(x)]_+ = \sum_x [\mu(x) - \pi(x)]_+,$$

where the equality follows because both distributions sum to one. Sampling from the rejection residual therefore contributes mass $[\mu(x) - \pi(x)]_+$ to token x. The two outcomes reconstruct the target mass coordinatewise,

$$\min\{\pi(x), \mu(x)\} + [\mu(x) - \pi(x)]_+ = \mu(x). \tag{12}$$

If a later sibling test replaces the immediate residual sample, the same identity applies because that later procedure returns the current residual distribution. Backward induction over the ordered siblings shows that the first accepted candidate, or the final residual sample when all candidates reject, has distribution p_u/Z_u. This statement averages over the candidate samples drawn during drafting; each proposal may depend on the earlier samples masked from that sibling set.

Tree routing. For binary accept bits, Equation 7 satisfies

$$s_u + \sum_j f_{u,j} = 1.$$

Consequently, every reached internal node either reaches exactly one first-accepted child or becomes the endpoint. Starting from $\mathrm{reach}_0 = 1$, the recurrence in Algorithm 3 therefore defines one reached path and exactly one endpoint bit z_u. A leaf has $s_u = 1$ and is the endpoint whenever it is reached. The sum $r_\star = \sum_u z_u \bar{r}_u$ selects the final residual of that unique endpoint, so the single call to HIERSAMPLE returns the required MSS bonus.

Conditioned on reaching node u, the local identity shows that the next emitted token follows the target distribution at that prefix. If a child is accepted, p_v is the target distribution conditioned on the extended prefix, and the same argument applies at v. If all children reject or a leaf is reached, the bonus closes the decoding round with the remaining part of the same target distribution. Induction over the reached depth therefore shows that the accepted path followed by the bonus has the target autoregressive law. The root itself is not emitted; if all root children reject, the decoding round contains only the root-residual bonus. This proves the theorem. □

Boundary convention. We use the strict comparison in Equation 5; equality has probability zero for continuous coins. Public padding entries are assigned accept bit zero. All computations still follow the public tree schedule, while encrypted reach and endpoint bits determine which results contribute to the output.