UC Santa Barbara

UC Santa Barbara Electronic Theses and Dissertations

Title

Stochastic Sampling for Sparse, Hardware-Efficient Inference and Optimization

Permalink

https://escholarship.org/uc/item/9v5434rb

ISBN

9798171612184

Author

Lee, Kyle

Publication Date

2026-09-07

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

University of California

Santa Barbara

Stochastic Sampling for Sparse, Hardware-Efficient Inference and Optimization

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy

in

Electrical and Computer Engineering

by

Kyle Lee

Committee in charge:

Professor Kerem Y. Camsari, Chair

Professor Li-C. Wang

Professor Zheng Zhang

Professor Yao Qin

September 2026

The Dissertation of Kyle Lee is approved.

Professor Li-C. Wang

Professor Zheng Zhang

Professor Yao Qin

Professor Kerem Y. Camsari, Committee Chair

June 2026

Stochastic Sampling for Sparse, Hardware-Efficient Inference and Optimization

Copyright © 2026

by

Kyle Lee

To my father, Owen Lee, whose unwavering support made this journey possible.

Acknowledgements

I am deeply grateful to my advisor, Professor Kerem Y. Camsari, for his guidance, trust, and intellectual generosity throughout my Ph.D. He gave me my first real chance in research when I was an inexperienced undergraduate, introduced me to the world of academia, and played a central role in my decision to pursue graduate study. I am grateful for his mentorship and for the encouragement he gave me to grow as a researcher.

I thank my committee members, Professors Li-C. Wang, Zheng Zhang, and Yao Qin for their time, feedback, and service on my committee.

I am grateful to my collaborators and coauthors, Shuvro Chowdhury, Corentin Delacour, Kevin Callahan-Coray, Kyle Jiang, Can Yaras, Samet Oymak, and Tathagata Srimani, for their technical discussions, insights, and contributions to this work. I also thank Abir Hasan, E. M. Hasantha Ekanayake, Nikhil Shukla, Xiuqi Zhang, Abdelrahman S. Abdelrahman, and Saleh Bunaiyan for collaboration on related publications and preprints listed in the curriculum vitae.

I thank the members of OPUS Lab and the broader UCSB research community for their discussion, support, and friendship throughout graduate school.

Finally, I thank my family and friends for their patience, encouragement, and support throughout graduate school.

Different portions of this dissertation were supported by different sources, as acknowledged below.

The c-bit work adapted in Chapter 2 and in the “Additional Details for Chaotic Bits” portion of Appendix A was partially supported by an Office of Naval Research Young Investigator

Program grant, a National Science Foundation CCF 2106260 grant, and the Army Research Office under grant number W911NF-24-1-0228.

Use was made of computational facilities purchased with funds from the National Science Foundation (CNS-1725797) and administered by the Center for Scientific Computing (CSC). The CSC is supported by the California NanoSystems Institute and the Materials Research Science and Engineering Center (MRSEC; NSF DMR 2308708) at UC Santa Barbara.

The following Army Research Office acknowledgment applies to the SANTA work adapted in Chapters 3–4 and to the "Additional Details for SANTA" portion of Appendix A.

Research was sponsored by the Army Research Office and was accomplished under Grant Number W911NF-24-1-0228. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

The following National Science Foundation Graduate Research Fellowship Program acknowledgment applies to work conducted while I was supported by the fellowship, including portions of the SANTA work adapted in Chapters 3–4 and the "Additional Details for SANTA" portion of Appendix A, as well as the stochastic-neuron work presented in Chapter 5 and the "Additional Details for Stochastic Neurons" section of Appendix A.

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. 2139319. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

The following National Research Platform acknowledgment applies to computational resources used for the SANTA and stochastic-neuron work in this dissertation.

This work used resources available through the National Research Platform (NRP) at the University of California, San Diego. NRP has been developed, and is supported in part, by

funding from National Science Foundation, from awards 1730158, 1540112, 1541349, 1826967, 2112167, 2100237, and 2120019, as well as additional funding from community partners.

Curriculum Vitæ

Kyle Lee

Education

2026 Ph.D. in Electrical and Computer Engineering, University of California, Santa Barbara.

2025 M.S. in Electrical and Computer Engineering, University of California, Santa Barbara.

2024 B.S. in Electrical Engineering, University of California, Santa Barbara.

Professional Appointments

2025 Graduate Student Researcher, January 2025–June 2026, Department of Electrical and Computer Engineering, University of California, Santa Barbara.

2024 Teaching Assistant, Fall 2024, Department of Electrical and Computer Engineering, University of California, Santa Barbara.

Publications and Preprints

2026 Abir Hasan, E. M. Hasantha Ekanayake, **Kyle Lee**, Kerem Y. Camsari, and Nikhil Shukla. "Breakdown of Gradient-Flow Dynamics in Oscillator Ising Machines from Harmonic Misalignment." *arXiv preprint*, 2026. `arXiv:2605.17243`.

2026 Corentin Delacour, Xiuqi Zhang, Abdelrahman S. Abdelrahman, Saleh Bunaiyan, **Kyle Lee**, Shuvro Chowdhury, and Kerem Y. Camsari. "Probabilistic Algorithms for Ising Machines: From Optimization to Generative AI." *Nature Reviews Electrical Engineering*, in press, 2026.

2026 **Kyle Lee**, Corentin Delacour, Kevin Callahan-Coray, Kyle Jiang, Can Yaras, Samet Oymak, Tathagata Srimani, and Kerem Y. Camsari. "Stochastic Sparse Attention for Memory-Bound Inference." In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026. `arXiv:2605.01910`.

2026 Kevin Callahan-Coray, **Kyle Lee**, Kyle Jiang, and Kerem Y. Camsari. "Restoring Sparsity in Potts Machines via Mean-Field Constraints." *npj Unconventional Computing*, 3, 50, 2026. `doi:10.1038/s44335-026-00097-x`.

2025 **Kyle Lee**, Shuvro Chowdhury, and Kerem Y. Camsari. "Noise-Augmented Chaotic Ising Machines for Combinatorial Optimization and Sampling." *Communications Physics*, 8, 35, 2025. doi:10.1038/s42005-025-01945-1.

2025 Saleh Bunaiyan, Corentin Delacour, Shuvro Chowdhury, **Kyle Lee**, and Kerem Y. Camsari. "IsingFormer: Augmenting Parallel Tempering with Learned Proposals." *arXiv preprint*, 2025. arXiv:2509.23043.

Awards/Honors

2025 National Science Foundation Graduate Research Fellow.

Fields of Study

Electrical and Computer Engineering; probabilistic computing, combinatorial optimization, and machine learning. Research supervised by Professor Kerem Y. Camsari.

Abstract

Stochastic Sampling for Sparse, Hardware-Efficient Inference and Optimization

by

Kyle Lee

Modern workloads increasingly rely on combinatorial optimization and large-scale neural generation, yet their performance is often limited less by arithmetic than by memory movement, energy cost, and the difficulty of exploring large discrete state spaces. This dissertation advances stochastic sampling as a hardware-aware computing principle, using controlled stochasticity to replace portions of dense deterministic computation with sparse, scalable alternatives.

First, the dissertation develops noise-augmented chaotic bit networks for Ising machine optimization and sampling. Chaotic bits use deterministic billiard-ball dynamics to generate binary spin states, providing a chaotic analogue of probabilistic bits. Chaotic bit networks reproduce key sampling phenomena, including agreement with the quantum Boltzmann law in a one-dimensional transverse-field Ising model and critical dynamics in classical Ising and spin-glass systems. The dissertation further introduces a hybrid chaotic-probabilistic adaptive parallel tempering algorithm, improving performance on spin-glass optimization and semiprime factorization while retaining the asynchronous, massively parallel structure that motivates specialized Ising machine hardware.

Building on this sampling perspective, the dissertation addresses the memory bandwidth bottlenecks of transformer inference. In long-context autoregressive decoding, repeatedly streaming the key-value cache often dominates execution time. To address this, the dissertation

introduces Stochastic Additive No-mulT Attention (SANTA), which samples value rows from the post-softmax attention distribution and aggregates them using sparse gather-and-add operations. SANTA provides an unbiased estimator of the dense post-softmax value aggregation, reducing value-cache access while removing value-stage multiplications. Variance-reduced sampling and custom GPU kernels yield practical long-context acceleration, including up to 1.5x kernel-level attention speedup and up to 1.25x end-to-end decode-latency speedup for Llama-3.1-8B-Instruct while preserving accuracy on reasoning and long-context benchmarks. A complementary Bernoulli score-stage estimator suggests a path toward sparse key access.

Finally, the dissertation extends this framework to transformer feedforward networks, demonstrating how stochastic activations can dynamically induce sparsity that may be exploitable by sparse down-projection kernels. Collectively, these contributions establish stochastic sampling as a practical route toward hardware-efficient inference and optimization, from chaotic Ising machines to sparse transformer inference.

Contents

Curriculum Vitae **viii**

Abstract **x**

1 Introduction **1**

1.1 Motivation 1

1.2 Overview of Contributions 3

1.3 Organization of the Dissertation 4

1.4 Permissions and Attributions 6

2 Noise-Augmented Chaotic Ising Machines for Combinatorial Optimization and Sampling **8**

2.1 Probabilistic Bits, Chaotic Bits, and Boltzmann Sampling 8

2.2 Chaotic-Bit Dynamics, Asynchrony, and Implicit Noise 14

2.3 Sampling and Critical-Dynamics Benchmarks 18

2.3.1 Quantum Boltzmann Sampling in a 1D Transverse-Field Ising Model 18

2.3.2 Critical Scaling Dynamics in Ising and Spin-Glass Systems 21

2.4 Noise-Augmented Adaptive Parallel Tempering 29

2.5 Combinatorial Optimization Benchmarks 32

2.5.1 3D Spin-Glass Optimization 32

2.5.2 Semiprime Factorization with Invertible Boolean Logic 34

2.6 Hardware Implications and Discussion 40
2.7 Chapter Summary 42

3 Stochastic Sparse Attention for Memory-Bound Inference 44
3.1 Long-Context Autoregressive Decoding and the KV Bottleneck 44
3.2 Stochastic Additive No-mulT Attention 47
3.3 Unbiasedness, Variance, and Concentration 49
3.4 Variance-Reduced Sampling 51
3.5 Theoretical Compute and Memory Accesses 55
3.6 Accuracy on Reasoning and Long-Context Benchmarks 58
3.6.1 GSM8K 59
3.6.2 MMLU 60
3.6.3 8k Long-Context Accuracy Checks 60

4 GPU Kernels for Long-Context Stochastic Sparse Attention 62
4.1 Parallelizing Stochastic Attention 63
4.2 Proportional Sample Allocation: S^2ANTA-Prop 64
4.3 Speculative Local Sampling: S^2ANTA-Flash 65
4.4 Decode-Step Kernel and 32k Accuracy Benchmarks 66
4.5 End-to-End Batched Decoding and Energy Considerations 69
4.6 Bernoulli $qK^\top$ Sampling for Sparse Key Access 71
4.6.1 Bernoulli $qK^\top$ Estimator and BitNet Results 71
4.6.2 Mean Group Query for GQA 72
4.6.3 Long-Context Results for Bernoulli $qK^\top$ 74
4.6.4 Combining Score-Stage and Value-Stage Sampling 74
4.7 Discussion, Limitations, and Hardware Outlook 75

5 Stochastic Neurons for Sparse Transformer Feedforward Networks 77
5.1 Motivation and Overview 77

5.2 Stochastic SiLU and SwiGLU Gates 79
5.3 Retrofitting a Pretrained Language Model with Stochastic Gates 83
5.4 Language-Model-Scale Trainability 88
5.5 Memory-Read Opportunity from Stochastic Zeros 90
5.6 Sparse FFN Kernel Opportunity 92
5.7 Discussion 94
5.8 Limitations 95
5.9 Conclusion 96

6 Conclusion and Outlook **97**
6.1 Summary of Contributions 97
6.2 Future Directions 98

A Additional Derivations and Experimental Details **101**
A.1 Additional Details for Chaotic Bits 101
A.1.1 Kibble–Zurek Scaling Derivation and Measurement Procedure 102
A.1.2 Adaptive Parallel Tempering Preprocessing 105
A.1.3 Time-to-Solution Metric for Factorization 106
A.2 Additional Details for SANTA 106
A.2.1 Formal Definition of SANTA 106
A.2.2 Unbiasedness and Variance Proofs 107
A.2.3 Formal Properties of Variance-Reduced Sampling 109
A.2.4 Concentration Bound for SANTA 111
A.2.5 Bernoulli qK^{T} Derivations 112
A.2.6 Empirical Variance and Unique-Key Measurements 114
A.2.7 Extended Accuracy Results and Top-k Comparisons 119
A.2.8 Extended LongBench v2 Results 123
A.2.9 Layer Sensitivity and Sample-Budget Ablations 124

A.2.10 Distributed RL Implementation Details 129
A.2.11 Evaluation Protocols 131
A.2.12 S^2ANTA-Prop Pseudocode 134
A.2.13 S^2ANTA-Flash Pseudocode 135
A.2.14 Decode-Step Microbenchmark Protocol 137
A.2.15 Additional Kernel Accuracy Results 140
A.2.16 HumanEval Code-Generation Results 140
A.2.17 Same-State Fidelity and Tile-Sensitivity Analyses 141
A.3 Additional Details for Stochastic Neurons 144
A.3.1 Qualitative GSM8K Generations 144
A.3.2 Qwen2.5-0.5B GSM8K LoRA Protocol 146
A.3.3 FineWeb-Edu 707M Pretraining Protocol 147
A.3.4 Sparse FFN Runtime Benchmark Protocol 148

Bibliography **149**

List of Figures

2.1 **Overview of probabilistic bits and chaotic bits.** (a) A probabilistic bit (p-bit) contains a random number generator with uniform distribution between -1 and 1. $\tanh(I_i)$ is a threshold for latching m_i. (b) A chaotic bit (c-bit) features deterministic billiard dynamics with a tunable slope. The billiard is periodic, given that I_i is held constant. When the billiard reaches -1 or $+1$, m_i is latched, and the billiard changes direction. (c) p-bits and c-bits form similar, asynchronous architectures, employing the same synaptic function. (d) A 20-bit factorizer conceptualized with p-bits, represented as a graph with 310 nodes and 1200 edges. (e) The 3D spin glass problem, with coupling strengths $J_{ij} \in \{-1, +1\}$. (f) The adaptive parallel tempering (APT) algorithm. Replicas of the same network run at different inverse temperature β in parallel. Swap attempts are made between pairs of replicas with the closest β, according to the Metropolis criterion. 9

2.2 **Oscillatory behavior of parallel updates.** (a) Distribution of spin configurations for a 5 p-bit full adder, obtained analytically from the Boltzmann distribution. (b) Analytical distribution for p-bits employing parallel updates, as opposed to sequential Gibbs sampling [1]. (c) Numerical results for parallel p-bit updates, collected over 10^5 Monte Carlo sweeps. (d) c-bits exhibit similar pathological behavior when their phases x_i and states m_i are all initialized to the same value, $+1$. Samples are collected over 10^5 time steps. 16

2.3 **Chaotic bits emulate a 1D transverse field Ising model.** Chaotic bits emulate a 1D ferromagnetic chain ($J_{ij} = +2$) of 8 qubits described by the quantum transverse Ising Hamiltonian (Equation 2.9). For emulation purposes, we use Suzuki-Trotter decomposition with $R = 250$ replicas. We show average magnetization as a function of the transverse field (Γ_x) when the system is at constant inverse temperature $\beta = 10$. A symmetry breaking magnetic field of $h_i = 1$ is applied in the $+\hat{z}$ direction such that when $\Gamma_x = 0$, all spins point in the $+\hat{z}$ direction. Red dots show the results of 80 independent trials of c-bit simulations, and a single set of spin configurations is recorded at the end of $t_a = 1000$ time steps, for each trial. Blue triangles show the average numerical result. The green dashed line is an analytical solution from solving Equation 2.13 as a function of Γ_x. 20

2.4 **Critical scaling dynamics on the 2D Ising and 3D spin glass models.** (a) Residual energy density at the critical point (ρ_E^c) as a function of simulated annealing (SA) duration for a 50×50 ferromagnetic Ising model lattice with nonperiodic boundary conditions ($J_{ij} = +1$). We perform linear regression on the 5 leftmost points of each data series to avoid finite-size effects. Error bars show 95% bootstrap confidence intervals. Purple squares represent p-bits, yellow triangles show c-bits ($\Delta t = 0.1$), and green circles show c-bits ($\Delta t = 0.01$). Time is measured in Monte Carlo sweeps for p-bits and time is dimensionless for c-bits. The inset plot shows short-term exponential divergence of c-bit systems with slightly perturbed initial conditions. The slope represents a positive (maximum) Lyapunov exponent, implying chaotic behavior. (b) ρ_E^c vs. annealing duration for cubic spin-glass instances ($L = 11$, $J_{ij} \in \{-1, +1\}$). (c) ρ_E^c as a function of inverse annealing velocity for the 2D ferromagnetic Ising model, using p-bits. Different annealing schedules are shown in a data collapse by considering the quench rate at the critical point, α. (d) ρ_E^c vs. inverse annealing velocity for the 2D ferromagnetic Ising model, using c-bits. 24

2.5 **Optimization of the 3D spin glass problem.** Residual energy density (ρ_E^f) as a function of total computational time (Nt_a) for cubic spin-glass instances with side length $L = 7$ ($J_{ij} \in \{-1, +1\}$). Curves scale as a power law: $\rho_E^f \propto t_a^{-\kappa_f}$. Time is measured in Monte Carlo sweeps for p-bits and time is dimensionless for c-bits. Error bars show 95% bootstrap confidence intervals. Results include p-bit simulated annealing (SA, purple squares), probabilistic adaptive parallel tempering (APT, blue circles), c-bit SA (yellow triangles), and c-bit APT with stochasticity (orange diamonds). 33

2.6 **Semiprime factorization with adaptive parallel tempering.** (a) p-bits are utilized to construct AND gates and full adders, that are the fundamental gates for a multiplier. The product bits are clamped to a semiprime number as in invertible Boolean logic [2]. (b) Energy profile of the 5 coldest replicas during hybrid chaotic-probabilistic adaptive parallel tempering (APT). $E - E_0$ represents distance from ground state energy. (c) β profile obtained from APT preprocessing. (d) Time-to-solution vs. product size using a 10-bit × 10-bit factorizer for semiprime numbers. p-bit simulated annealing (SA, purple squares) yields a slope of 0.441, while p-bit APT (blue circles) yields a slope of 0.438. c-bit SA (yellow triangles) exhibits a slope of 0.658, and hybrid APT (orange diamonds) has a slope of 0.420. (e) Average replica swap probabilities from factoring a specific semiprime number over 100 trials (p-bit APT shown in blue, hybrid APT shown in orange). 36

3.1 SANTA requires $S \ll n_k$ memory accesses to sampled rows of V, replacing the dense post-softmax weighted value sum with sparse gather-and-add. The remaining normalization by $1/S$ is a bit shift if S is a power of 2. 47

4.1 S^2ANTA-prop kernel overview. 64

4.2 S^2ANTA-flash kernel overview. 65

4.3 S^2ANTA kernel latency on Llama 8B tensor shapes for 1 decoding step. (a) S^2ANTA-prop demonstrates a 1.50× speedup relative to FlashInfer at a 32k-token context. (b) S^2ANTA-flash similarly exhibits a 1.51× speedup. Both operating points ($S = 128$ for prop, $S = 2048$ for flash) correspond to configurations that recover baseline accuracy. 67

4.4 Bernoulli $qK^\top$ represents query magnitudes as Bernoulli probabilities and carries the sign in a sparse ternary query vector, sparsifying along the feature dimension d_k with multiplier-free arithmetic. 71

5.1 **Expectation-matched stochastic SiLU creates exact zeros.** The stochastic activation $\tilde{a} = xB$, with $B \sim \text{Bernoulli}(\sigma(x))$, matches deterministic SiLU $x\sigma(x)$ in pointwise expectation while producing exact zeros with probability $1 - \sigma(x)$. In a transformer FFN, zero intermediate channels expose down-projection weight reads that can in principle be skipped. 80

5.2 **Adapter capacity recovers generated-answer accuracy under stochastic gates.** With stochastic SwiGLU gates enabled, FFN-only LoRA improves with rank but remains below the deterministic no-LoRA base. Adapting both FFN and attention projections improves recovery, with rank-256 FFN+attention LoRA reaching $13.44 \pm 0.52\%$ GSM8K generated-answer exact match compared with 13.12% for the deterministic base. Error bars show standard deviation over 10 stochastic activation-mask repeats. 87

5.3 **Layerwise sparsity in the best Qwen stochastic LoRA setting.** The rank-256 FFN+attention checkpoint produces exact zeros throughout the transformer stack during a fixed gold-solution teacher-forced pass on GSM8K. The overall zero-activation rate is 58.69%, matching the sparsity value used for the memory-read estimate. Layerwise zero rates range from 41.37% to 69.19%. 87

5.4 **From-scratch pretraining remains stable under stochastic SiLU.** A 707M-parameter GPT-style model trained with stochastic SiLU + STE remains stable over 2B tokens, although it converges to higher validation CE and lower next-token accuracy than the deterministic SiLU baseline. 89

A.1 Mean L2 norm error for Bernoulli $qK^\top$. The score error is calculated during decoding with $d_k = 128$, a context length $n_k = 1024$, and averaged over 100 instances $q \sim \mathcal{N}(0,1)$, $K \sim \mathcal{N}(0,1)/\sqrt{d}_k$, $V \sim \mathcal{N}(0,1)$. 113

A.2 Empirical variance of SANTA on single-hop QA prompts (8k tokens). 115

A.3 Ablations with one-hot stochastic attention. Baseline scores employ a fixed sampling budget of 16 and 256 for DeepSeek 7B and Llama 8B models, respectively. The horizontal axis shows the index of the model layer which is reduced to stochastic hard attention. 124
A.4 (a) Learned schedule with a total budget of 224 samples across all 28 transformer blocks (DeepSeek 7B). (b) RL iterations over time. 128
A.5 Kubernetes deployment schematic. 130

List of Tables

3.1 Per-query, per-head value-stage compute and memory for dense scaled dot-product attention (SDPA) and SANTA. n_k is the number of keys, d_k is the head dimension, and S is the SANTA sample budget per query. Value reads count V elements accessed per query; writes count output elements. 55

3.2 Unique V rows read, measured as the union over all query positions in a causal prefill pass and reported as a percentage of all n_k rows in the value matrix. Each entry is the mean over 10 trials and all attention heads, with Gaussian inputs $Q, K, V \sim \mathcal{N}(0, 1)$ (FP16), H=32 heads, and head dimension d_k=128. 56

3.3 Per-head FLOPs and memory for scaled dot-product attention (SDPA), top-k, and SANTA. n_q: number of queries, n_k: number of keys, d_k: head dimension, k: top-k budget, S: SANTA sample budget per query. "Reads" count logical value/key elements accessed; "Writes" count output or score elements. Softmax, top-k selection, and sampling overheads are omitted. 57

3.4 GSM8K accuracy and average context length (prompt + answer) for Llama-3.1-8B-Instruct. S: SANTA sample budget. Accuracy shows 95% bootstrap confidence intervals. 59

3.5 MMLU accuracy and average context length (prompt + answer) for Llama-3.1-8B-Instruct. S: SANTA sample budget. Accuracy shows 95% bootstrap confidence intervals. 60

3.6 Accuracy of Llama 8B on 4 long-context tasks with an 8k-token prompt. Tasks: frequent word extraction (FWE), needle-in-a-haystack (NIAH), single-hop QA (QA_1), multi-hop QA (QA_2). S: SANTA sample budget. Accuracy shows 95% bootstrap confidence intervals. 61

4.1 Accuracy of Llama 8B on 4 long-context tasks with a 32,768-token prompt. SDPA is used in prefill. Tasks: frequent word extraction (FWE), needle-in-a-haystack (NIAH), single-hop QA (QA_1), multi-hop QA (QA_2). S: sample budget. Accuracy shows 95% bootstrap confidence intervals. 68

4.2 LongBench v2 accuracy for Llama 8B at the main accuracy-matching operating points. S: sample budget. Accuracy shows 95% bootstrap confidence intervals. 68

4.3 End-to-end batched generation latency for Llama-3.1-8B-Instruct on RTX 6000 Ada. Prompts are truncated to 30k tokens, batch size is 6, and each prompt decodes 2048 new tokens. Prefill remains dense; S^2ANTA is applied only during decode. Results are averaged over 9 batches. 69

4.4 Performance of Bernoulli $qK^\top$ (stratified) on GSM8K (BitNet 2B) with the test set evaluated three times, and confidence intervals obtained by bootstrapping. 73

4.5 Accuracy of Llama 8B on 4 long-context tasks with a 4,096-token prompt. Tasks: frequent word extraction (FWE), needle-in-a-haystack (NIAH), single-hop QA (QA_1), multi-hop QA (QA_2). B: Bernoulli $qK^\top$ (mean group query) sample budget. Accuracy shows 95% bootstrap confidence intervals. 74

4.6 Performance of Bernoulli $qK^\top$ (mean group query) + SANTA on GSM8K (BitNet 2B) with the test set evaluated three times, and confidence intervals computed by bootstrapping. 75

5.1 **Qwen2.5-0.5B GSM8K LoRA retrofit under stochastic SwiGLU gates.** Generated-answer exact match is the primary task metric on the 1,319-example GSM8K test split and is reported as mean ± standard deviation over 10 stochastic activation-mask repeats. CE/PPL and zero activations are auxiliary teacher-forced diagnostics on the official gold solutions. 85

5.2 **707M-parameter GPT-style pretraining on FineWeb-Edu.** Both models use 12 layers, width 2048, FFN dimension 8192, sequence length 512, tied embeddings, and 2.000B training tokens. CE, PPL, and top-1 next-token prediction accuracy are computed on 25M-token validation and test sets from the final checkpoints. 90

5.3 **Conservative theoretical memory-read opportunity from exact stochastic zeros.** The projections that produce the mask remain dense; only the final FFN projection is counted as skippable when an intermediate channel is zero. These are weight-read opportunities, not end-to-end latency predictions. 92

5.4 **FFN-level sparse down-projection benchmark.** For the measured 12-FFN path, consuming the stochastic hidden vector sparsely reduces time from 1.502066 to 1.147353 ms/token, a 1.309× speedup relative to the dense stochastic FFN implementation. The timed path includes the up projection and stochastic activation in both cases; the dense-versus-sparse difference is how the zero-containing hidden vector is consumed by the down projection. This is an FFN-level result only, not an end-to-end model speedup. 93

A.1 Unique-key count vs. S (8192 tokens, single-query). 115

A.2 GSM8K accuracy and average context length (prompt + answer) for DeepSeek-R1-Distill-Qwen-7B. k: number of keys in top-k, S: SANTA sample budget. Accuracy shows 95% bootstrap confidence intervals. 119

A.3 GSM8K accuracy and average context length (prompt + answer) for Llama-3.1-8B-Instruct. 119

A.4 MMLU accuracy and average context length (prompt + answer) for DeepSeek-R1-Distill-Qwen-7B. k: number of keys in top-k, S: SANTA sample budget. Accuracy shows 95% bootstrap confidence intervals. 120
A.5 MMLU accuracy and average context length (prompt + answer) for Llama-3.1-8B-Instruct. 120
A.6 Accuracy of Llama 8B on 4 long context tasks with an 8k-token prompt. Tasks: frequent word extraction (FWE), needle-in-a-haystack (NIAH), single-hop QA (QA_1), multi-hop QA (QA_2). k: number of keys for top-k. S: SANTA sample budget. Accuracy shows 95% bootstrap CIs. 121
A.7 Full LongBench v2 sweep for Llama 8B. S: sample budget. Accuracy shows 95% bootstrap confidence intervals. 123
A.8 RL sample-schedule performance. Accuracy shows 95% bootstrap confidence intervals. 129
A.9 Prompting and grading for DeepSeek 7B on GSM8K. 131
A.10 Prompting and grading for DeepSeek 7B on MMLU. 132
A.11 Prompting and grading for Llama 8B on GSM8K. 132
A.12 Prompting and grading for Llama 8B on MMLU. 133
A.13 Prompting and grading for Llama 8B on RULER prompts. 133
A.14 Accuracy of Llama 8B on 4 long-context tasks with an 8k-token prompt. Tasks: frequent word extraction (FWE), needle-in-a-haystack (NIAH), single-hop QA (QA_1), multi-hop QA (QA_2). S: sample budget. Accuracy shows 95% bootstrap confidence intervals. 140
A.15 HumanEval pass@1 accuracy for Llama 8B. S: sample budget. Pass@1 shows 95% bootstrap confidence intervals. 141

A.16 Same-state numerical fidelity of S^2ANTA-flash and S^2ANTA-prop on 32k-token FWE and QA_2 prompts. Relative L2 error and cosine similarity compare final-layer sparse and dense attention outputs AV at matched decode states. Top-1 agreement reports whether a full one-token approximate forward pass preserves the dense model's top-1 next-token prediction. 142
A.17 Sensitivity of S^2ANTA-flash to attention support spread across tiles. Tile size is fixed to 256 and sample budget is fixed to $S = 2048$. "Tiles to reach 90% attention mass" is computed from the dense attention distribution at each evaluated decode state. 143
A.18 Sensitivity of S^2ANTA-flash to tile size at 32k-token contexts. Sample budget is fixed to $S = 2048$. Smaller tiles imply more total tiles and more opportunity for sample waste under uniform per-tile sampling. 143

Chapter 1

Introduction

1.1 Motivation

A central motivation of this dissertation is that stochasticity can be used as a computational resource rather than only as a source of error. In Ising machines, the relevant object is an exponentially large space of 2^N spin configurations, and useful algorithms must explore only a tiny fraction of this space while still finding low-energy states or representative samples. In long-context transformer attention, the analogous bottleneck is not a state space but memory traffic: SANTA samples a small set of value-cache rows rather than aggregating over every value row at every decode step. In transformer feedforward networks, stochastic gates sample neuron or channel activity so that exact zeros expose down-projection weight reads that can be skipped. Across these settings, the dissertation studies stochastic sampling as a hardware-conscious way to replace dense deterministic work with controlled sparse selection.

As Moore's law stagnates, there is growing interest in unconventional computing schemes and hardware accelerators. Among these emerging technologies are Ising machines [3], employing devices such as probabilistic bits (p-bits) [2, 4, 5, 1], coupled oscillators [6, 7, 8, 9, 10, 11, 12], photonic devices [13, 14, 15], spin-wave devices [16], and other nonlinear elements [17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28]. Ising machines have traditionally focused on providing scal-

ing and prefactor improvements in solving combinatorial optimization problems [29, 30, 31]. Efforts to link Ising machines to generative machine learning [32, 33, 34, 35, 36] and quantum simulation problems are also underway [37, 38, 39].

The first part of this dissertation evaluates and improves an earlier Ising machine concept based on chaotic bits (c-bits), originally proposed in Ref. [40]. Chaotic bits resemble coupled oscillators, following a set of deterministic neuron update rules without explicit randomness. However, unlike oscillator-based Ising machines that encode the Ising spin in the continuous phase of oscillators, c-bits make use of oscillating billiard balls that set a latch to a $+1$ or -1 state. As such, chaotic bits do not require cumbersome subharmonic injection locking schemes to binarize naturally continuous phase variables. The potential hardware implementation of chaotic Ising machines without explicit random number generators, while retaining similarity to the mathematics of p-bits, is appealing. At the same time, deterministic c-bits cannot straightforwardly employ powerful Monte Carlo algorithms such as adaptive parallel tempering, which offer better algorithmic scaling for Circuit SAT and 3D spin glass problems. This motivates the noise-augmented c-bit approach developed in Chapter 2, where deterministic c-bits are augmented with stochasticity for improved performance in combinatorial optimization and sampling.

The second part of this dissertation turns to stochastic sparsity in transformer inference. The transformer architecture [41] has given rise to modern language models [42, 43, 44]. For long context LLM inference, throughput during autoregressive decoding is often limited by memory bandwidth: generating each new token repeatedly streams the Key–Value (KV) cache for all prior tokens. For example, for Llama-3.1-8B-Instruct at a 32k-token context, streaming bf16 keys and values is approximately 128 MB *per layer per generated token*, and KV memory access cost scales linearly with context length and batch size.

This bandwidth bottleneck motivates SANTA, short for Stochastic Additive No-mulT Attention, developed in Chapters 3 and 4. Instead of permanently discarding cache contents, SANTA reduces the amount of value-cache data accessed during the decode-time softmax–V

aggregation. Its primary focus is the post-softmax value aggregation, where dense attention computes AV by multiplying attention weights by all n_k value rows. SANTA samples $S \ll n_k$ indices from the post-softmax distribution and aggregates only those sampled rows of V via gather-and-add. This yields an unbiased estimator of the post-softmax value aggregation while reducing value-cache row reads per query from n_k to S and replacing the dense value-stage weighted multiplications with sampled additive accumulation, up to the fixed normalization by $1/S$. Variance-reduced variants, denoted S^2ANTA, use stratified and systematic sampling to improve accuracy while admitting GPU-friendly implementations.

1.2 Overview of Contributions

The first contribution of this dissertation is a noise-augmented chaotic-bit framework for combinatorial optimization and probabilistic sampling. Chapter 2 refines deterministic c-bits by introducing noise, improving performance in combinatorial optimization and achieving algorithmic scaling comparable to probabilistic bits. It tests c-bits on three problem classes: a quantum problem, criticality in 3D spin glasses, and hard combinatorial optimization in the form of integer factorization with invertible logic gates. The chapter shows that c-bits agree closely with the quantum Boltzmann law in a 1D transverse-field Ising model and exhibit qualitatively similar critical dynamics to p-bits in 2D Ising and 3D spin-glass models. It further proposes a noise-augmented c-bit approach via adaptive parallel tempering, which outperforms fully deterministic c-bits running simulated annealing. This suggests that mixing stochasticity with deterministic c-bits can provide a useful hybrid computing scheme for asynchronous, massively parallel hardware implementations.

The second, and central LLM-systems contribution, is SANTA, a stochastic sparse attention method for memory-bound long-context inference. Chapters 3 and 4 introduce SANTA and variance-reduced S^2ANTA as unbiased estimators of the post-softmax value aggregation that reduce value-cache reads from n_k to S. The dissertation develops the estimator, its variance-

reduced sampling variants, and GPU kernels that parallelize sampling and sparse accumulation. On Llama-3.1-8B-Instruct at 32k-token contexts, the S^2ANTA CUDA kernels match baseline accuracy while achieving up to $1.5\times$ decode-step attention-kernel speedup over FlashInfer and FlashDecoding on an NVIDIA RTX 6000 Ada. In batched long-context generation, these kernel gains translate to up to $1.25\times$ end-to-end decode-latency speedup. The SANTA chapters also characterize tunable accuracy–efficiency trade-offs on GSM8K, MMLU, and long-context benchmarks.

Chapter 4 also includes Bernoulli $qK^\top$ sampling as a secondary score-stage extension. This mechanism represents query elements as Bernoulli variables to form sparse ternary queries in $\{-1, 0, +1\}$, yielding an unbiased estimator of $qK^\top$ and inducing feature-sparse key access during decoding. In this dissertation, however, the implemented kernels and measured speedups are driven primarily by SANTA's value-stage sparsification. Bernoulli $qK^\top$ is therefore presented as a complementary path toward sparse key access, not as the main source of the LLM acceleration results.

A final, forward-looking component of the dissertation is the study of stochastic neurons for sparse transformer feedforward networks in Chapter 5. This chapter extends the same broad sampling principle from value-cache rows to FFN channels: stochastic gates decide which intermediate neurons are active for a token, and exact zero channels expose down-projection weight reads that may be skipped. This work is kept separate from the c-bit and SANTA contributions because it is unpublished dissertation work and does not yet claim end-to-end serving acceleration. Its role in the dissertation is to identify an FFN-side opportunity that could complement the main SANTA attention results in future systems.

1.3 Organization of the Dissertation

Chapter 2 presents noise-augmented chaotic Ising machines for combinatorial optimization and sampling. It introduces probabilistic bits and chaotic bits, explains the deterministic c-bit

dynamics, and evaluates whether c-bit networks can reproduce the useful sampling behavior associated with p-bit networks. The chapter then studies quantum Boltzmann sampling, critical dynamics, and hard optimization benchmarks, with particular emphasis on why added stochasticity and adaptive parallel tempering improve the performance of otherwise deterministic c-bit systems.

Chapter 3 introduces Stochastic Additive No-mulT Attention for memory-bound inference. It begins from the long-context KV-cache bottleneck in autoregressive decoding, defines the SANTA estimator for post-softmax value aggregation, and develops the main unbiasedness, variance, and concentration arguments. It also introduces the stratified and systematic sampling variants that form S^2ANTA and evaluates the accuracy–efficiency trade-offs on reasoning and long-context benchmarks.

Chapter 4 focuses on GPU kernels and systems-level evaluation for long-context stochastic sparse attention. It explains how SANTA-style sampling can be parallelized on GPUs, develops the S^2ANTA-prop and S^2ANTA-flash kernel strategies, and reports decode-step kernel timing, 32k-token accuracy benchmarks, and end-to-end batched decoding results. The chapter also presents Bernoulli $qK^\top$ sampling as a complementary, forward-looking method for sparse key access.

Chapter 5 studies stochastic neurons for sparse transformer feedforward networks. It defines expectation-matched stochastic SiLU/SwiGLU gates, evaluates LoRA-based adaptation of Qwen2.5-0.5B on GSM8K, reports a moderate-scale FineWeb-Edu pretraining experiment, and measures whether the resulting exact zeros can be exploited by sparse FFN down-projection kernels.

Chapter 6 concludes the dissertation by summarizing the main contributions and outlining future directions for hardware-aware stochastic inference, sparse kernels, and probabilistic computing. It returns to the common theme that useful stochasticity can reduce the number of states, memory rows, or neuron channels touched by an algorithm while preserving useful optimization, sampling, or inference behavior.

1.4 Permissions and Attributions

1. Chapter 2 and the "Additional Details for Chaotic Bits" portion of Appendix A are adapted from: Kyle Lee, Shuvro Chowdhury, and Kerem Y. Camsari, "Noise-augmented Chaotic Ising Machines for Combinatorial Optimization and Sampling," *Communications Physics* 8, 35 (2025). doi:10.1038/s42005-025-01945-1. This article is licensed under a Creative Commons Attribution 4.0 International License (https://creativecommons.org/licenses/by/4.0/), which permits adaptation and reproduction with appropriate credit. The material has been adapted and edited for dissertation format and chapter organization. Kyle Lee and Kerem Y. Camsari conceived the idea for the study; Kerem Y. Camsari supervised the study; Kyle Lee conducted the numerical experiments with assistance from Shuvro Chowdhury; Kyle Lee wrote the initial manuscript draft; and all authors participated in writing the manuscript, analyzing the experiments, and discussing the results.

2. Chapters 3–4 and the "Additional Details for SANTA" portion of Appendix A are adapted from: Kyle Lee, Corentin Delacour, Kevin Callahan-Coray, Kyle Jiang, Can Yaras, Samet Oymak, Tathagata Srimani, and Kerem Y. Camsari, "Stochastic Sparse Attention for Memory-Bound Inference," In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. This work is licensed under a Creative Commons Attribution 4.0 International License (https://creativecommons.org/licenses/by/4.0/). The material has been adapted and edited for dissertation format and chapter organization. Kyle Lee's contributions included leading the project, designing the SANTA algorithm, running numerical experiments for SANTA, developing and benchmarking GPU kernels, and drafting the manuscript.

3. Chapter 5 and Appendix A.3 contain original, unpublished dissertation work conducted by Kyle Lee at the University of California, Santa Barbara. This material is not adapted from a prior publication or preprint. The empirical results, figures, and tables in this chapter

and appendix were generated for this dissertation project; external models, datasets, software, and prior work are cited in the relevant chapter, appendix, and bibliography.

Chapter 2

Noise-Augmented Chaotic Ising Machines for Combinatorial Optimization and Sampling

2.1 Probabilistic Bits, Chaotic Bits, and Boltzmann Sampling

Figure 2.1 gives an overview of the probabilistic-bit and chaotic-bit primitives used in this chapter, along with representative problem classes and algorithms considered later in the chapter.

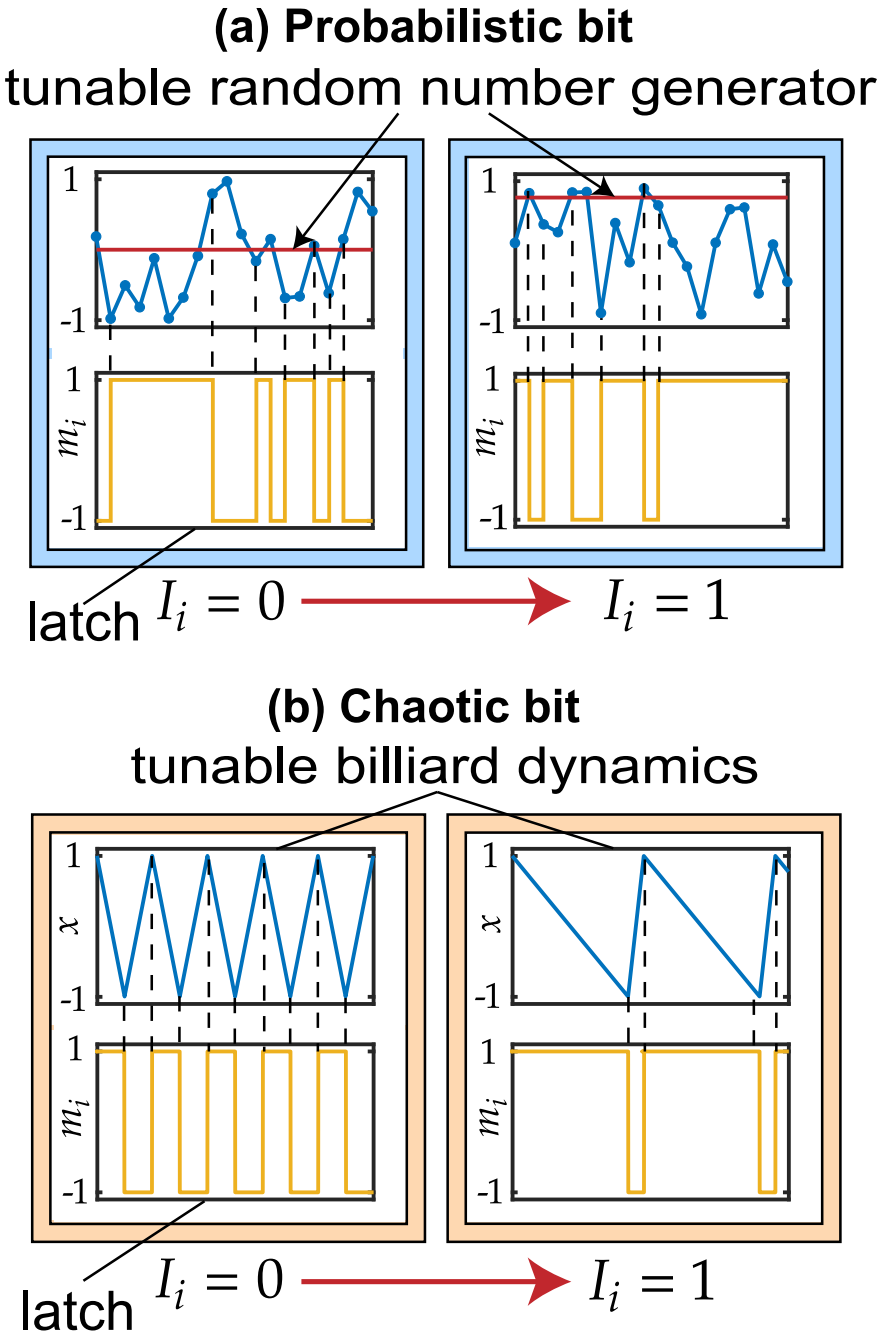


Figure 2.1: **Overview of probabilistic bits and chaotic bits.** (a) A probabilistic bit (p-bit) contains a random number generator with uniform distribution between -1 and 1. $\tanh(I_i)$ is a threshold for latching m_i. (b) A chaotic bit (c-bit) features deterministic billiard dynamics with a tunable slope. The billiard is periodic, given that I_i is held constant. When the billiard reaches -1 or $+1$, m_i is latched, and the billiard changes direction.

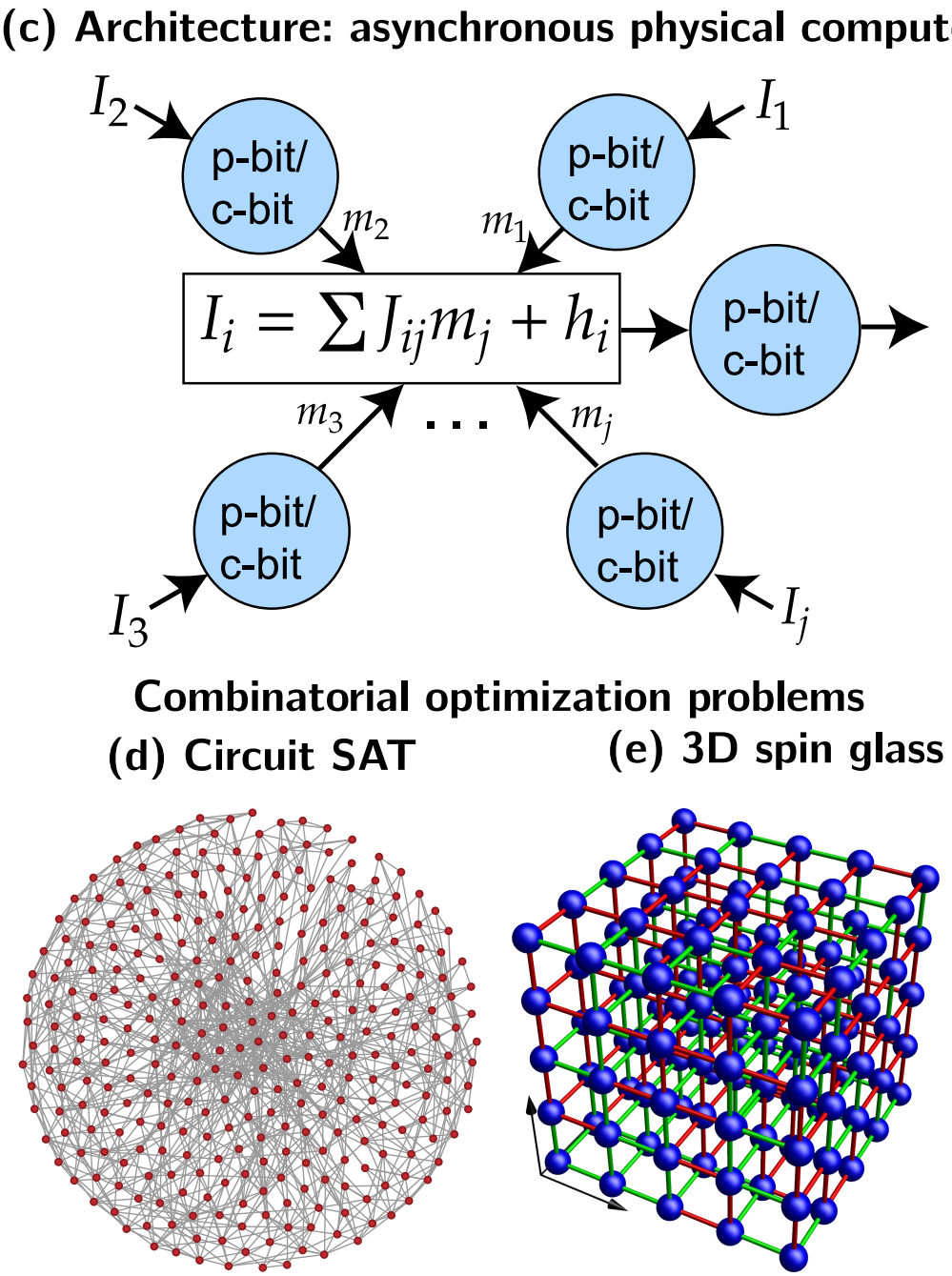


Figure 2.1: **Overview of probabilistic bits and chaotic bits (continued).** (c) p-bits and c-bits form similar, asynchronous architectures, employing the same synaptic function. (d) A 20-bit factorizer conceptualized with p-bits, represented as a graph with 310 nodes and 1200 edges. (e) The 3D spin glass problem, with coupling strengths $J_{ij} \in \{-1, +1\}$.

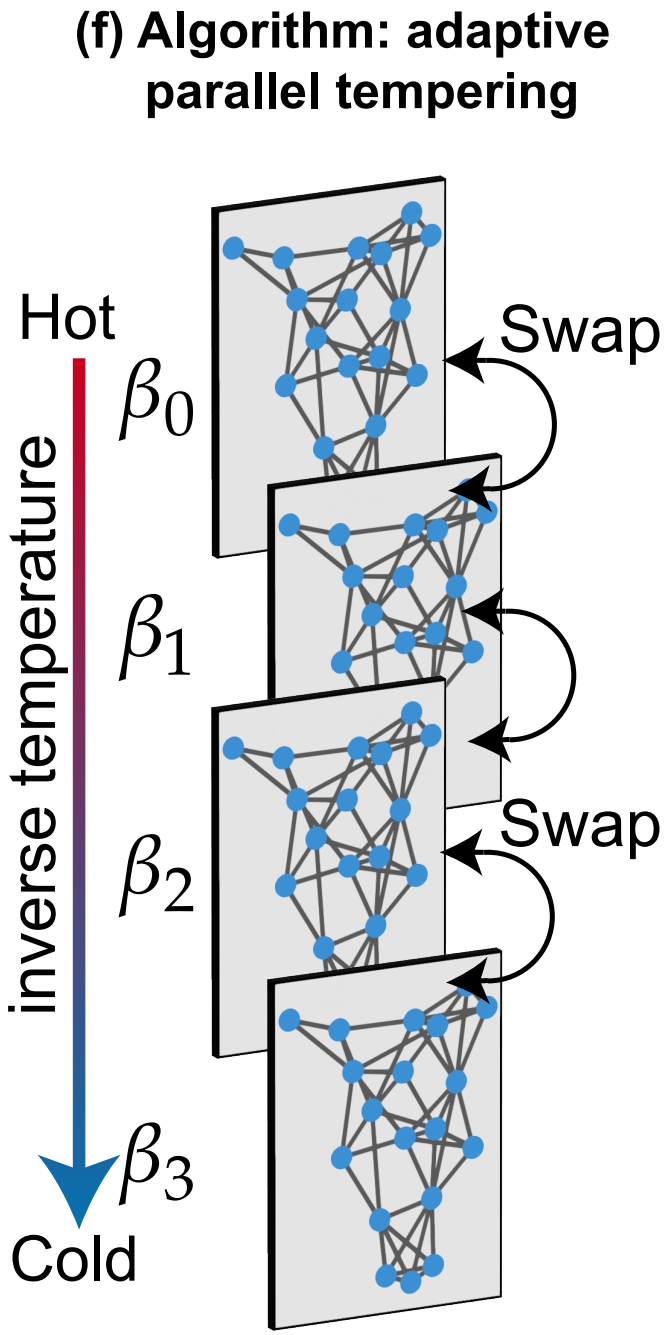


Figure 2.1: **Overview of probabilistic bits and chaotic bits (continued).** (f) The adaptive parallel tempering (APT) algorithm. Replicas of the same network run at different inverse temperature β in parallel. Swap attempts are made between pairs of replicas with the closest β, according to the Metropolis criterion.

Probabilistic computers perform a discrete Markov Chain Monte Carlo (MCMC) algorithm, simulated in software via Gibbs sampling (Fig. 2.1c). p-bit networks are typically described by second-order interactions in energy and stochastic neurons:

$$m_i = \text{sgn}[\tanh(\beta I_i) - \text{rand}_u(-1, 1)] \tag{2.1}$$

$$I_i = \sum_j J_{ij} m_j + h_i \tag{2.2}$$

where $m_i \in \{-1, +1\}$ and $\text{rand}_u(-1, 1)$ is a random uniform distribution between -1 and 1 (Fig. 2.1a, c). β is inverse temperature. J_{ij} and h_i represent weights and biases. At equilibrium, p-bit networks sample from the Boltzmann distribution, given by:

$$p(\{m\}) = \frac{1}{Z} \exp[-\beta E(\{m\})] \tag{2.3}$$

$$E(\{m\}) = -\frac{1}{2} \sum_{i,j} J_{ij} m_i m_j - \sum_i h_i m_i \tag{2.4}$$

where $\{m\}$ is a spin configuration, Z is the partition function, and $E(\{m\})$ is the energy of a spin configuration.

Chaotic bits follow deterministic update rules, inspired by p-bit equations, which model an oscillating billiard ball and a latch. We reformulate the original chaotic bit equations [40, 45] to emphasize their similarity to p-bits:

$$\begin{aligned} \frac{dx_i}{dt} &= -m_i + \tanh(\beta I_i) \\ x_i = +1 &\implies \text{set } m_i \text{ to } +1 \\ x_i = -1 &\implies \text{set } m_i \text{ to } -1 \end{aligned} \tag{2.5}$$

where x_i represents the position of a billiard between the boundaries $x_i = -1$ and $x_i = +1$. The billiard has a tunable slope, dx_i/dt, which is dependent on the input I_i. When the billiard x_i reaches -1 or $+1$, the c-bit state m_i is latched to that value, and the billiard changes direction (Fig. 2.1b). Note that in numerical or hardware implementations of c-bits, the billiard state x_i

in Eq. 2.5 can slightly exceed $+1$ or -1 due to discretization errors.

The c-bit definition can be motivated from the Boltzmann distribution, analogous to p-bits and Gibbs sampling. Consider a single spin m_i with a fixed set of neighbors. For p-bits, combining Eq. 2.2 and Eq. 2.3, the conditional probability of m_i being ± 1 is expressed as:

$$\begin{aligned} p(m_i = +1) &= \frac{1}{Z} \exp\left[\beta(I_i - E_{\text{rest}})\right] \\ p(m_i = -1) &= \frac{1}{Z} \exp\left[\beta(-I_i - E_{\text{rest}})\right] \end{aligned} \tag{2.6}$$

where Z is the partition function and E_{rest} is the part of the global energy, or the environment, that is not dependent on p-bit i. When expressed as a ratio of probabilities, Eq. 2.6 becomes:

$$\frac{p(m_i = +1)}{p(m_i = -1)} = \exp\left(2\beta I_i\right) \tag{2.7}$$

which describes the behavior of a single p-bit given fixed neighbors.

As a deterministic unit, c-bit updates do not involve probabilities, but rather they try to achieve the same sampling ratio of $+1$'s to -1's by modifying the lifetime of up and down trajectories of the billiard ball, conditioned on the state of its neighbors (Fig. 2.1b):

$$\frac{\tau(m_i = +1)}{\tau(m_i = -1)} = \frac{\left|\frac{dx_i}{dt} \quad \text{when } m_i = -1\right|}{\left|\frac{dx_i}{dt} \quad \text{when } m_i = +1\right|} = \exp\left(2\beta I_i\right) \tag{2.8}$$

where τ is the lifetime of a c-bit at its current state ($m_i = \pm 1$). In other words, c-bits are defined such that the time average of the c-bit state $\langle m_i \rangle = \tanh\left(\beta I_i\right)$, analogous to a p-bit. The key point is that these lifetimes are deterministic, unlike a p-bit whose state is stochastically sampled at each update.

Theoretical work has shown convergence to the Boltzmann distribution for simple toy models of 2 c-bits [46]. However, it has not been determined that larger c-bit networks necessarily sample from the Boltzmann distribution as stochastic p-bits do. Experimental work suggests that c-bit networks seem to closely approximate the Boltzmann distribution for 2D Ising models

of small sizes [45]. Furthermore, it has been shown that c-bits match stochastic p-bits for equilibrium statistics on the 2D ferromagnetic Ising model [40] and the Potts model [45]. c-bits have been used to determine the critical temperature (T_c) and critical exponents (ν, β, and γ) of the Ising universality class [45]. Simulated annealing has been demonstrated with c-bits [40, 47]. Additionally, CMOS and analog VLSI implementations have been proposed [48, 49]. Nonetheless, the theoretical equivalence of p-bits to c-bits without explicit noise is far from clear. Moreover, the initial phase randomization and rounding errors in the solution of ODEs, or physical noise in real systems, might be responsible for this striking correspondence.

The comparison of p-bits (Eq. 2.1 and Eq. 2.2) with c-bits (Eq. 2.5 and Eq. 2.2) is fundamental. In the case of p-bits, the fundamental theorem of Markov Chains [50, 51] ensures that a network of p-bits updated sequentially via Gibbs sampling will eventually reach the Boltzmann distribution, defined by Eq. 2.3. Ultimately, this is what enables mapping problems of interest to p-bit networks and then solving them with algorithms such as simulated annealing and parallel tempering. Networks of c-bits do not have an analogous proof that they sample exactly from the Boltzmann distribution, due to the lack of explicitly probabilistic and serial updates.

2.2 Chaotic-Bit Dynamics, Asynchrony, and Implicit Noise

Timing and asynchrony are central to using c-bit networks as samplers. Note the following parallel to p-bits. It is well-known that in Gibbs sampling, simultaneously updating connected nodes leads to pathological oscillations [52, 53], preventing easy parallelization of the chain. For c-bits, it seems that the differential equation formulation in continuous time prevents simultaneous updates naturally, as long as the time difference between subsequent billiard ball arrivals is greater than the time it takes to update a latch. When c-bits are discretized in software, latches are set instantaneously within a single time step, Δt, chosen to solve the differential

equations.

To better understand how critical serial updating is for c-bits, we construct a simple example, with 5 c-bits, or equivalently 5 p-bits, interconnected such that their low energy states represent the truth table of a full adder, given by the interactions [54, 55]:

$$J_{FA} = \begin{bmatrix} 0 & -1 & -1 & 1 & 2 \\ -1 & 0 & -1 & 1 & 2 \\ -1 & -1 & 0 & 1 & 2 \\ 1 & 1 & 1 & 0 & -2 \\ 2 & 2 & 2 & -2 & 0 \end{bmatrix}$$

with no bias terms ($h = 0$). The columns correspond to carry in, A, B, sum, and carry out bits, respectively. Fig. 2.2a shows the analytically obtained Boltzmann distribution from Eq. 2.3 at $\beta = 0.5$, which can be numerically approximated by sequential p-bit updates.

Fig. 2.2b shows the probability distribution of 32 possible spin configurations in a 5 p-bit full adder when p-bits are updated in parallel rather than sequentially. The steady-state distribution corresponding to this Markov chain can be obtained analytically [1, 51] and describes the numerical sampling shown in Fig. 2.2c well.

Fig. 2.2d shows that similar pathological oscillations between states 3 and 28 are observed in c-bit networks when the initial states are synchronized by making all x_i and m_i equal in the beginning. These oscillations go away if the initial phases are sufficiently separate and the integration step, Δt, is small. We conclude that c-bit networks implicitly rely on sequential updating for correct operation, despite their continuous-time and massively parallel dynamics. While simultaneous c-bit updates in an integration step Δt may be considered an artifact of software discretization, hardware implementations will have similar restrictions due to finite setup/hold times and synapse delays associated with calculating I_i. The hardware implications of this update mechanism are revisited later in this chapter.

P-bits and c-bits have different notions of time. There is no clear correspondence between a

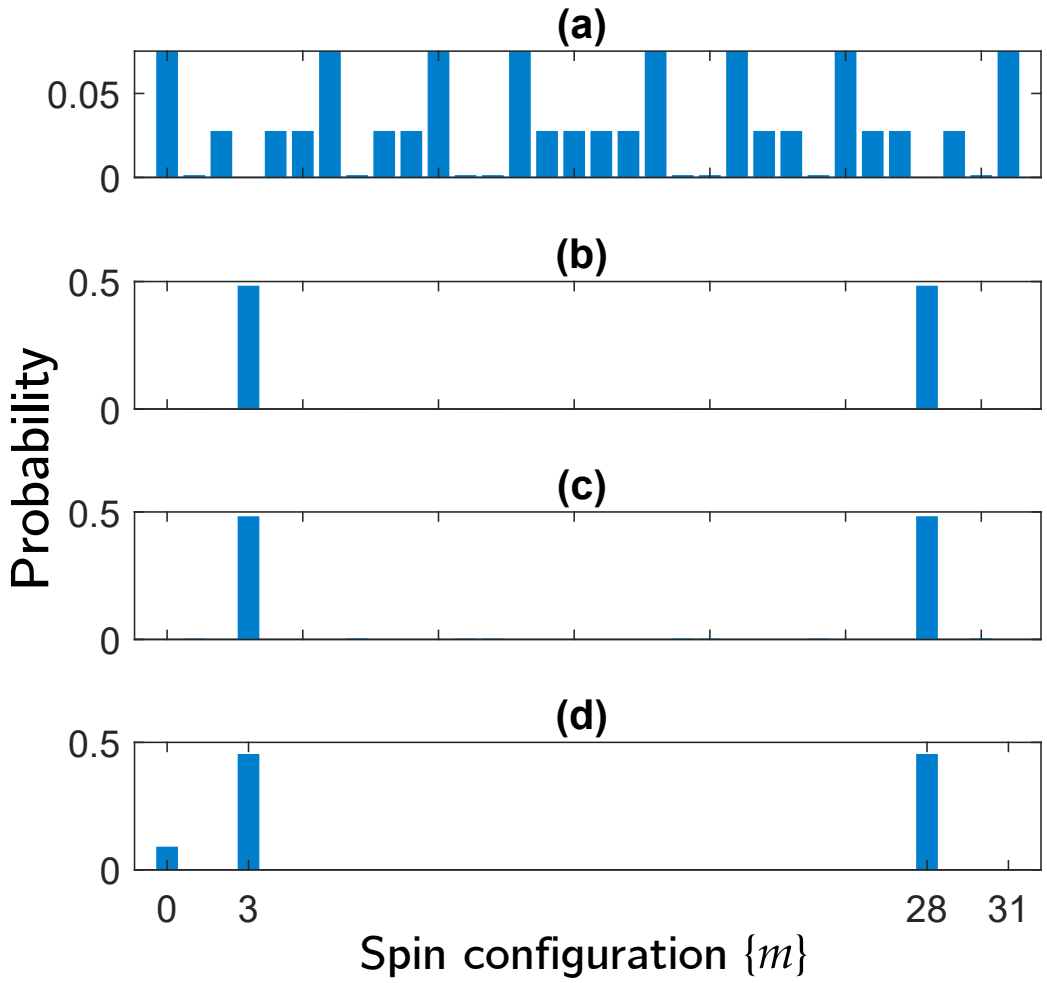


Figure 2.2: **Oscillatory behavior of parallel updates.** (a) Distribution of spin configurations for a 5 p-bit full adder, obtained analytically from the Boltzmann distribution. (b) Analytical distribution for p-bits employing parallel updates, as opposed to sequential Gibbs sampling [1]. (c) Numerical results for parallel p-bit updates, collected over 10^5 Monte Carlo sweeps. (d) c-bits exhibit similar pathological behavior when their phases x_i and states m_i are all initialized to the same value, $+1$. Samples are collected over 10^5 time steps.

p-bit Monte Carlo sweep (MCS), in which Eqs. 2.1 and 2.2 are sequentially solved for each p-bit in the network, and the time variable t of a c-bit network. In this chapter, we sample the state $\{m\}$ of a continuously updating c-bit network at natural numbers $t = \{0, 1, 2, \ldots, t_a\}$. Because of these different notions of time, in order to draw comparisons between p-bit and c-bit algorithms, we employ power-law and exponential scaling arguments, which hold independent of prefactors. Furthermore, we apply similar algorithmic parameters to both p-bits and c-bits to ensure fair comparisons.

Most existing work employs a definition of c-bit billiard velocity (dx_i/dt) that grows exponentially at cold temperatures ($\beta \gg 1$), with the exception of one previous study [45]. This is problematic because optimization algorithms such as simulated annealing and adaptive parallel tempering require cold temperatures, which may result in unbounded slopes. In this chapter, we employ a c-bit definition that has a maximum billiard speed of 2, see Eq. 2.5, suitable for software and hardware implementation. Unless otherwise specified, we employ Euler's method with a step of $\Delta t = 0.1$ to simulate the dynamics of coupled c-bits.

Our bounded c-bit choice is also a potential improvement over previous hardware implementations, which may be suffering from accuracy losses due to exponentially growing slopes. For example, the anomalous increase of energy plots in a prior FPGA implementation is likely due to the overflows and underflows in slopes [47].

Finally, in all of our experiments, the initial phases of the billiard balls and the states of the latches are randomized. Together, the asynchronous dynamics of the billiard balls and this phase randomization ensure nonsimultaneous updates with quasi-random arrivals. In addition, rounding errors and the choice of a naive integration scheme provide additional noise in our simulations. These effects appear to be key reasons behind the near-equivalence of c-bits to p-bits, where explicit and implicit inclusion of noise plays a central role. This timing distinction motivates the sampling and critical-dynamics benchmarks that follow, where p-bit and c-bit comparisons are made through matched algorithmic settings and scaling behavior rather than through a direct one-to-one mapping between MCS and continuous time.

2.3 Sampling and Critical-Dynamics Benchmarks

2.3.1 Quantum Boltzmann Sampling in a 1D Transverse-Field Ising Model

As a difficult sampling problem that requires high-quality pseudorandom number generators [56], we consider a 1D ferromagnetic chain of 8 qubits at constant inverse temperature $\beta = 10$. This system is described by the transverse Ising Hamiltonian in 1D,

$$H_{\mathrm{Q}} = -\sum_{i<j} J_{ij}\sigma_i^z\sigma_j^z - \sum_i h_i\sigma_i^z - \Gamma_x\sum_i \sigma_i^x \tag{2.9}$$

where σ_i^x and σ_i^z are Pauli spin matrices at site i. Note that each of these is a $2^N \times 2^N$ matrix and is computed as $\sigma_i^\alpha = I_2^{\otimes i-1} \otimes \sigma^\alpha \otimes I_2^{\otimes N-i} = I_2 \otimes ... \otimes \sigma^\alpha \otimes ... \otimes I_2$ with $\alpha \in \{x, z\}$ where $\otimes$ denotes the 'Kronecker product', I_2 is the 2×2 identity matrix, σ^z and σ^x are Pauli spin matrices only at the i-th term of the product:

$$\sigma^z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad \text{and} \quad \sigma^x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \tag{2.10}$$

The interaction weight $J_{ij} = +2$ for nearest neighbors with a periodic boundary condition and $J_{ij} = 0$ otherwise. $h_i = +1$, representing a symmetry breaking magnetic field in the $+\hat{z}$ direction. The Suzuki-Trotter transformation [37, 57] maps the 1D quantum Hamiltonian to the following 2D classical Hamiltonian:

$$H_{\mathrm{C}} = -\sum_{k=1}^{R}\left[\sum_{i<j}\frac{J_{ij}}{R}m_{i,k}m_{j,k} + \sum_i \frac{h_i}{R}m_{i,k} + \sum_i J_\perp m_{i,k}m_{i,k+1}\right] \tag{2.11}$$

where $J_\perp = -1/2\beta \ln\tanh(\beta\Gamma_x/R)$. R represents the total number of system replicas after

applying the quantum-to-classical mapping. Note the unusual dependence of this coupling to the inverse temperature, β.

For an infinite number of replicas ($R \to \infty$), this quantum-to-classical mapping is exact. In practice, the error scales as $\mathcal{O}(1/R^2)$ for a given β. Here, we employ 250 replicas, thus our system contains $N = 8$ qubits and $R = 250$ replicas, totaling 2000 classical spins. We simulate our classical system using c-bits. As we vary the value of the transverse field Γ_x, we measure the average magnetization:

$$\langle m_z \rangle = \frac{1}{NR} \sum_{k=1}^{R} \sum_{i=1}^{N} m_{i,k} \tag{2.12}$$

In theory, the system should obey the quantum Boltzmann Law:

$$\langle m_z \rangle = \frac{\mathrm{tr}\left[\exp\left(-\beta H_{\mathrm{Q}}\right) \sum_i \sigma_i^z\right]}{\mathrm{tr}\left[\exp\left(-\beta H_{\mathrm{Q}}\right)\right]} \tag{2.13}$$

$\sum_i \sigma_i^z$ in the numerator is the corresponding quantum operator for measuring spins along the z-direction. Notice that the argument of the 'exp $(\cdot)$' in Eq. 2.13 is a matrix. Evaluating this equation becomes intractable for large N, as it involves taking the exponential of a $2^N \times 2^N$ matrix. When N is small, as in this example, one can compute the exponential of a matrix directly by explicitly writing down H_{Q} and calculating its exponential. Also, to obtain numerically stable results at low temperatures, corresponding to high β, we first diagonalize the Hamiltonian and subtract the ground-state energy from the diagonals, which does not affect any observable quantities.

Figure 2.3 shows that numerical results from c-bits exhibit strong agreement with the quantum Boltzmann law, obtained by solving Eq. 2.13 as a function of Γ_x. A similar result has been shown in a previous work using stochastic p-bits [58]. We observe that c-bits appear to show comparable performance to p-bits, despite the fact that c-bits only use explicit randomization for the initialization of spins and phases at the start of each simulation.

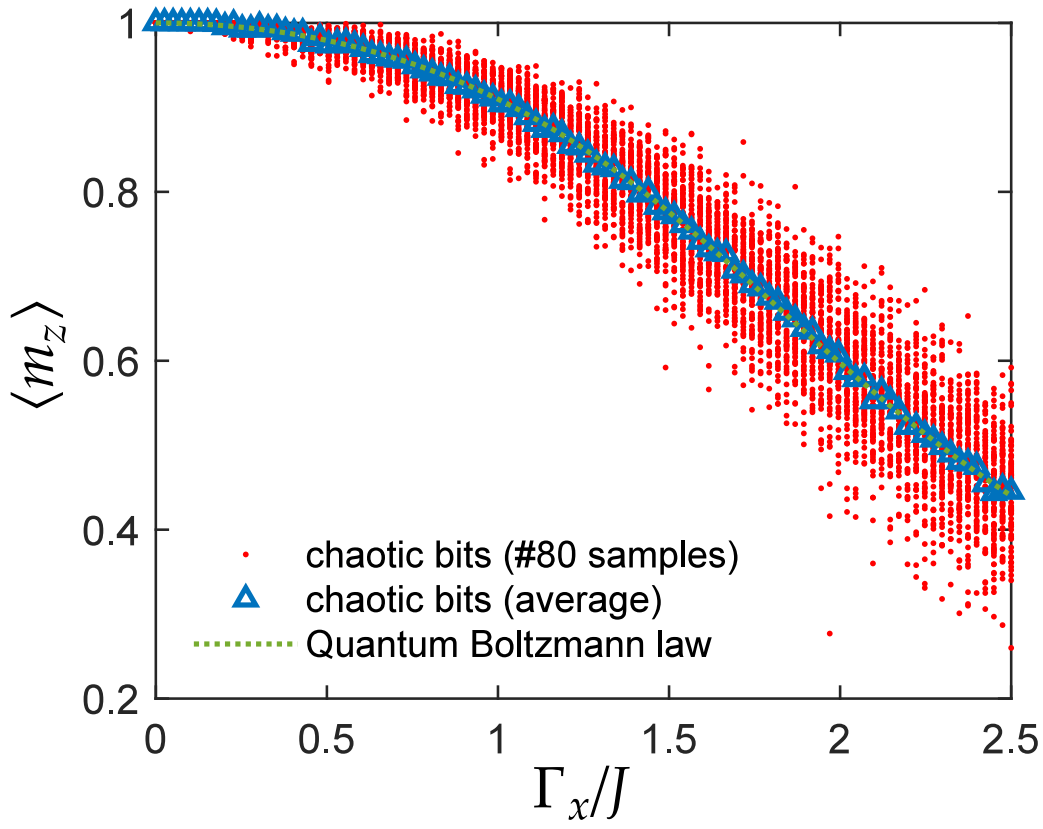


Figure 2.3: **Chaotic bits emulate a 1D transverse field Ising model.** Chaotic bits emulate a 1D ferromagnetic chain ($J_{ij} = +2$) of 8 qubits described by the quantum transverse Ising Hamiltonian (Eq. 2.9). For emulation purposes, we use Suzuki-Trotter decomposition with $R = 250$ replicas. We show average magnetization as a function of the transverse field (Γ_x) when the system is at constant inverse temperature $\beta = 10$. A symmetry breaking magnetic field of $h_i = 1$ is applied in the $+\hat{z}$ direction such that when $\Gamma_x = 0$, all spins point in the $+\hat{z}$ direction. Red dots show the results of 80 independent trials of c-bit simulations, and a single set of spin configurations is recorded at the end of $t_a = 1000$ time steps, for each trial. Blue triangles show the average numerical result. The green dashed line is an analytical solution from solving Eq. 2.13 as a function of Γ_x.

2.3.2 Critical Scaling Dynamics in Ising and Spin-Glass Systems

This subsection discusses critical dynamics of c-bit networks for the 2D Ising model and the 3D spin glass problem. The idea is to quench c-bit networks at varying anneal rates and measure the residual energy as a proxy of topological defects. The density of these defects is qualitatively predicted by the Kibble-Zurek Mechanism (KZM) [59], which relates equilibrium correlation lengths and relaxation times to defect densities borne out of non-equilibrium dynamics. In the present context, the study of KZM allows a comparison of c-bit performance with that of p-bits. In addition, residual energy may be a useful figure of merit for combinatorial optimization where scaling comparisons between different algorithms can be made [60]. The residual energy density at the critical point is defined as:

$$\rho_E^c = \frac{\langle E - E_c \rangle}{n} \tag{2.14}$$

where E_c is the equilibrium energy at the critical temperature and n is the number of spins in the system.

The equilibrium critical energies used to compute ρ_E^c are obtained from long constant-temperature p-bit runs; the detailed measurement procedure is given in Appendix A.

From the theory of phase transitions, phenomenological scaling arguments can be made to describe the residual energy in terms of an anneal with varying velocities. Briefly, the KZM picture starts from the divergence of equilibrium correlation lengths and relaxation times near a critical point, with $\xi(\lambda) \propto |\lambda_c - \lambda|^{-\nu}$ and $\tau(\lambda) \propto |\lambda_c - \lambda|^{-z\nu}$ for a tuning parameter λ, such as T or β. A general annealing schedule can be linearized near the critical point as $\lambda(t) = \lambda_c + \alpha(t - t_a) + O[(t - t_a)^2]$, and equating the remaining annealing time to the relaxation time gives a freeze-out scale that can be related to defect density by dimensional analysis. In this section, we do not attempt to reproduce critical exponents for c-bits or p-bits directly; rather, we use residual energy as an experimentally accessible proxy for defect density

and compare the scaling behavior of c-bits and p-bits. This gives the scaling relation

$$\rho_E^c \propto \alpha^{\kappa_c} \tag{2.15}$$

where κ_c is the exponent describing the scaling behavior of the residual energy density at the critical point, and α is the velocity of the anneal measured at the critical temperature for any schedule that starts from some initial temperature and ends at the critical temperature:

$$\alpha = \left.\frac{d\lambda(t)}{dt}\right|_{t=t_a}. \tag{2.16}$$

One way to make contact with known critical exponents and our numerically calculated exponents is to measure the number of defects via kinks [61] in Ising models, but our purpose here is to provide a comparative analysis with p-bits and c-bits, so direct verification of the exponents is not necessary.

First, we consider a 50×50 ferromagnetic Ising model lattice with nonperiodic boundary conditions ($J_{ij} = +1$). For a given annealing time t_a, β is geometrically increased from $\beta = 0.001$ to the critical temperature, $\beta = {T_c}^{-1}$. For this problem, we use the exact critical temperature calculated at the thermodynamic limit [62] as an approximate T_c for the finite-size model: $T_c = 2/\ln(1+\sqrt{2}) \approx 2.269$. When annealing terminates at the critical point, the energy of the spin configuration is recorded to calculate ρ_E^c. We average over 3000 randomized trials.

Although there is no obvious correspondence between a Monte Carlo sweep for behavioral models of p-bits and the time step of a c-bit, power-law exponents can be compared objectively since they do not depend on prefactors. Fig. 2.4a shows that c-bits and p-bits demonstrate similar critical scaling dynamics on the 2D ferromagnetic Ising model. Linear regression is performed on the 5 leftmost points in order to avoid finite-size effects. Both p-bit and c-bit curves exhibit downward concavity, but different time scales mean that this concavity does not manifest at the same time t_a for p-bits and c-bits. We provide the results of using Euler's

method in steps of $\Delta t = 0.1$ and $\Delta t = 0.01$. The c-bit curve using $\Delta t = 0.01$ appears to follow a similar power-law scaling, but with a steeper slope $|\kappa_c|$. The reason for this discrepancy is not clear. On the one hand, smaller Δt implies higher theoretical solver precision, while on the other hand, smaller Δt leads to more round-off errors due to double precision. It must also be noted that a smaller Δt requires more computational effort, and in hardware, Δt may have a lower bound dictated by the physical latch required to set the state of the oscillator.

Fig. 2.4b shows a similar experiment conducted on select instances of the 3D spin glass problem, studied by D-Wave [60]. We consider 300 different spin glass instances on a cubic lattice with side length $L = 11$ and $J_{ij} \in \{-1, +1\}$. Analogous to Fig. 2.4a, for a given annealing time t_a, we geometrically increase β to the inverse critical temperature $1/T_c$. For 3D spin glasses, $T_c \approx 1.1$ [63]. For each of 300 problem instances, we take an ensemble average over 40 random seeds. We again observe similar power-law scaling. Our p-bit results match the numerical results in the literature, with $\kappa_c \approx 0.51$ [60]. However, the values of κ_c do not seem to be similar when comparing between p-bits and c-bits. The Δt dependence of c-bit slopes also indicates that the underlying dynamics for c-bits might depend on separating near arrivals that latch a state, where Eq. 2.2 is always assumed to be infinitely fast. We conclude that while c-bits seem to obey a similar power law as p-bits, their dynamics seem different in subtle ways.

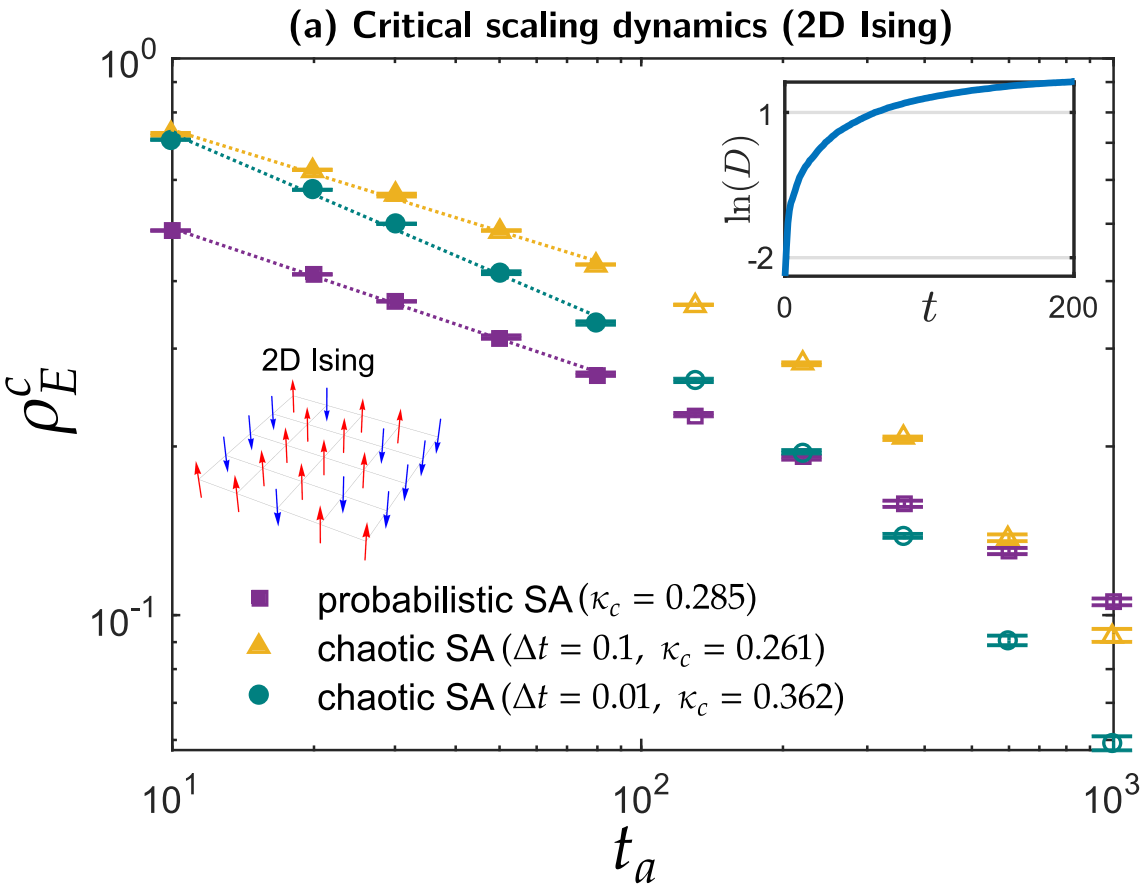


Figure 2.4: **Critical scaling dynamics on the 2D Ising and 3D spin glass models.** (a) Residual energy density at the critical point (ρ_E^c) as a function of simulated annealing (SA) duration for a 50×50 ferromagnetic Ising model lattice with nonperiodic boundary conditions ($J_{ij} = +1$). We perform linear regression on the 5 leftmost points of each data series to avoid finite-size effects. Error bars show 95% bootstrap confidence intervals. Purple squares represent p-bits, yellow triangles show c-bits ($\Delta t = 0.1$), and green circles show c-bits ($\Delta t = 0.01$). Time is measured in Monte Carlo sweeps for p-bits and time is dimensionless for c-bits. The inset plot shows short-term exponential divergence of c-bit systems with slightly perturbed initial conditions. The slope represents a positive (maximum) Lyapunov exponent, implying chaotic behavior.

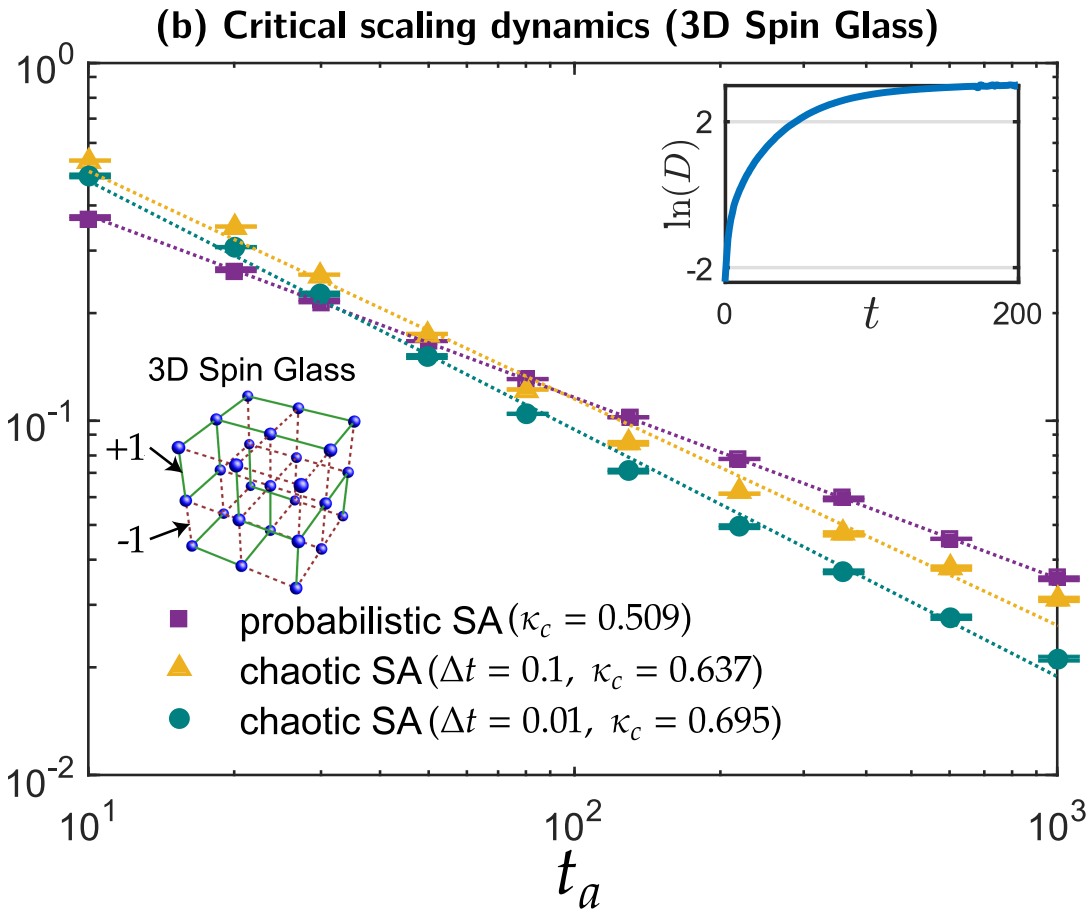


Figure 2.4: **Critical scaling dynamics on the 2D Ising and 3D spin glass models (continued).** (b) ρ_E^c vs. annealing duration for cubic spin-glass instances ($L = 11$, $J_{ij} \in \{-1, +1\}$).

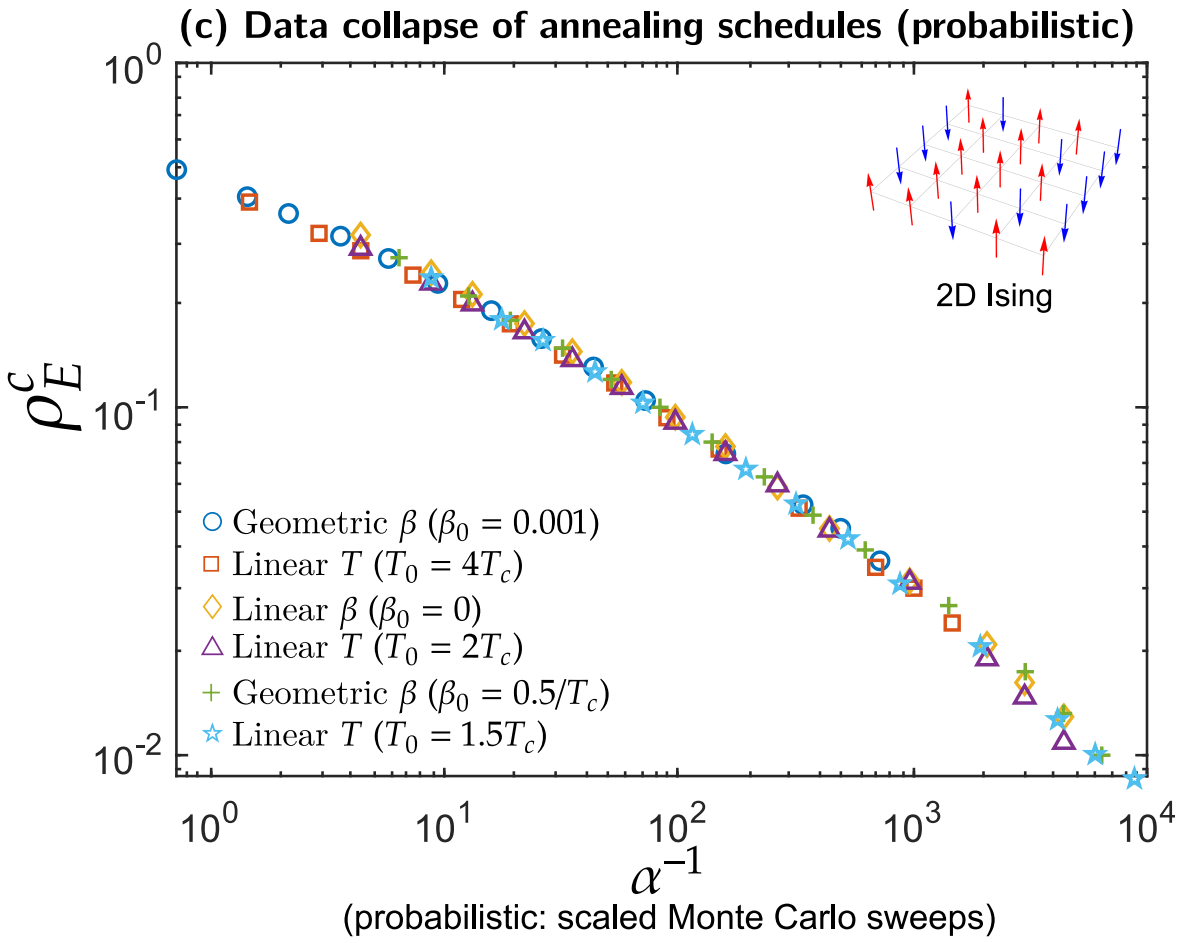


Figure 2.4: **Critical scaling dynamics on the 2D Ising and 3D spin glass models (continued).** (c) ρ_E^c as a function of inverse annealing velocity for the 2D ferromagnetic Ising model, using p-bits. Different annealing schedules are shown in a data collapse by considering the quench rate at the critical point, α.

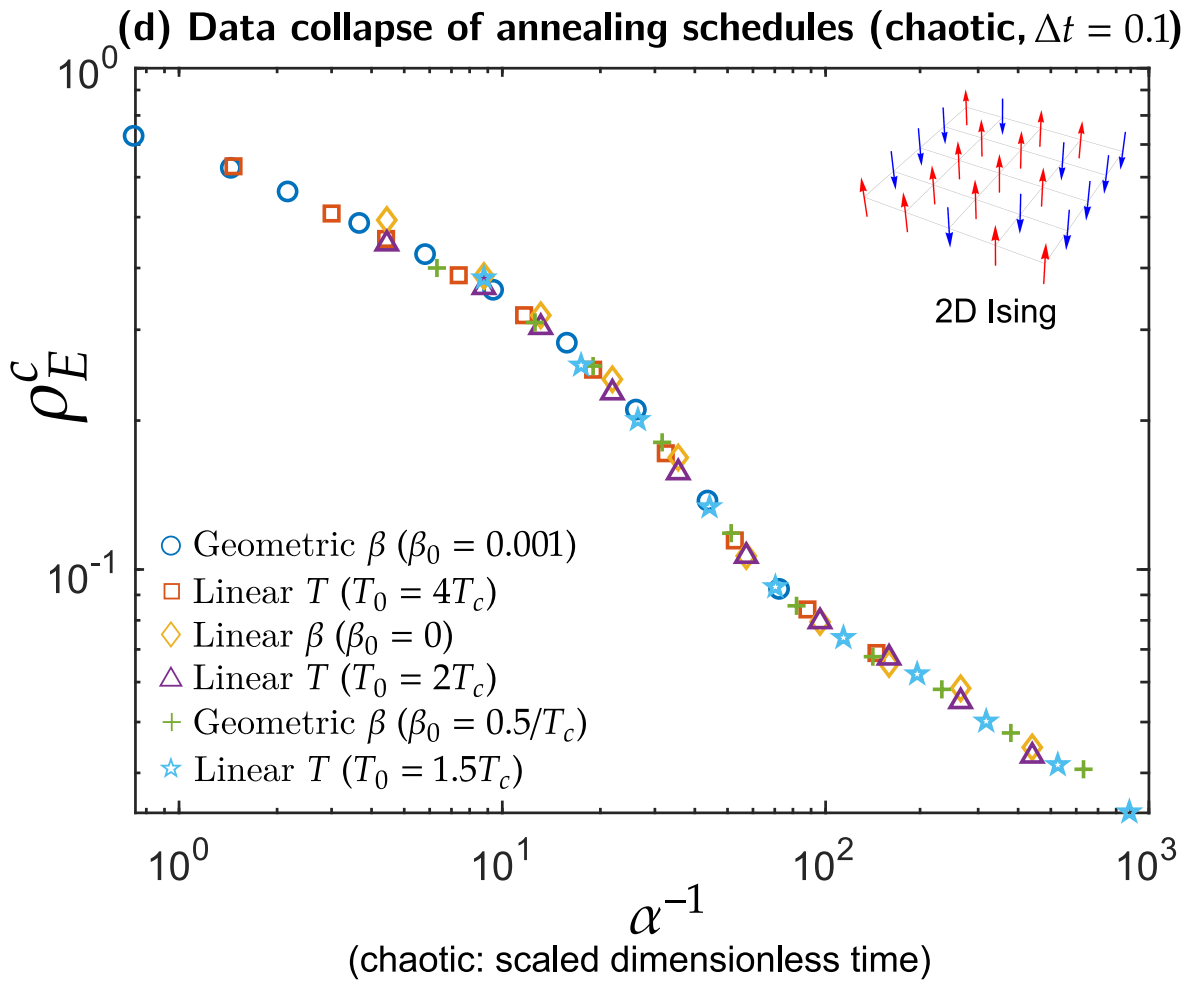


Figure 2.4: **Critical scaling dynamics on the 2D Ising and 3D spin glass models (continued).** (d) ρ_E^c vs. inverse annealing velocity for the 2D ferromagnetic Ising model, using c-bits.

One feature of chaotic dynamics is sensitivity to initial conditions. This can be tested by calculating Lyapunov exponents, which involves measuring the distance between a chosen initial condition and a slightly perturbed version of it as a function of time. If the distance in phase space grows exponentially in time, this is an indication of chaotic dynamics. In general, there is a spectrum of Lyapunov exponents, but the dynamics are dominated by the maximum Lyapunov exponent, which can be empirically measured. In the inset plots of Fig. 2.4a and Fig. 2.4b, we show maximum Lyapunov exponents for the 2D Ising model and the 3D spin glass problem, respectively. We observe that in both cases the log distance acquires a linear slope and then saturates. The linear and positive slope indicates chaotic dynamics before saturating. The saturation of the distance is indicative of the finite size of the phase space where a distance between two trajectories has an upper bound.

The specifics of the calculation are as follows: the phases $\mathbf{x}$ and the states $\mathbf{m}$ are randomly initialized. We then introduce a perturbation in the initial conditions by offsetting the phase x_i of a randomly selected c-bit by a magnitude of 0.1. We then consider the Euclidean distance between the phases of the perturbed and unperturbed systems: $D = \|\mathbf{x} - \mathbf{x}'\|$, where $\mathbf{x}'$ represents the perturbed system's phases. The log Euclidean distance is plotted over 200 time steps, with the system at constant inverse temperature $\beta = 1$. The slope represents the Lyapunov exponent λ, as $D \sim \exp \lambda t$. The results are averaged over 100 trials, where each trial compares one unperturbed system $\mathbf{x}$ with 10 perturbed versions, $\mathbf{x}'$.

In Fig. 2.4c–d, we show how both c-bits and p-bits exhibit a clear power-law relationship with critical exponents by comparing different annealing schedules with different initial temperatures. As Eq. 2.15 predicts, the simulations should exhibit universal behavior if plotted as a function of how fast the anneal is performed near the critical point. When plotted as a function of annealing velocity, α in Eq. 2.16, all c-bit and p-bit plots fall on top of each other, demonstrating an excellent collapse. The different exponents of c-bits and p-bits and the time-step dependence of the c-bit networks indicate that c-bit networks may not be a drop-in replacement for p-bit networks whose steady state provably samples from the Boltzmann distribution [51].

2.4 Noise-Augmented Adaptive Parallel Tempering

The noise augmentation in this section is motivated by a limitation of fully deterministic c-bit dynamics: c-bits without any randomness cannot straightforwardly employ the most powerful Monte Carlo algorithms, such as Adaptive Parallel Tempering, which offer better algorithmic scaling for the Circuit SAT and 3D spin glass problems. In the c-bit APT scheme below, deterministic c-bit updates are retained inside each replica, while stochasticity is introduced through probabilistic replica swaps.

Critical scaling dynamics are intriguing from a physics perspective, but it is not essential that c-bits obey the same physics as p-bits in an optimization context. For optimization, this chapter uses an adaptive version of the parallel tempering algorithm, abbreviated as APT [64, 65, 66]. A given p-bit or c-bit network is duplicated into N system replicas, with each replica at a different inverse temperature β (Fig. 2.1f). In fixed time intervals, corresponding to the time steps per swap attempt, a probabilistic swap attempt is conducted between adjacent replica pairs according to the Metropolis criterion:

$$P(\text{swap}) = \min[1, \exp(-\Delta\beta\Delta E)] \tag{2.17}$$

If this condition is met, then the adjacent replicas swap their spin states $\{m\}$. Alternatively, replicas may swap their temperature values with a rearrangement of replica indices, which may be more convenient in dedicated hardware implementations. Intuitively, hot replicas, corresponding to small β, explore the state space of possible spin states $\{m\}$. The Metropolis criterion makes low energy configurations more likely to swap to cold replicas, corresponding to high β. Over time, the coldest system replica tends toward the ground energy, E_0.

The APT algorithm can be applied using either p-bits or c-bits. While the neuron update rule is stochastic for p-bits (Eq. 2.1), it is deterministic for c-bits (Eq. 2.5). Replica swaps via the Metropolis criterion are always probabilistic (Eq. 2.17). Therefore, while p-bit APT is a purely probabilistic algorithm, c-bit APT is a hybrid chaotic-probabilistic scheme, where neuron

updates are deterministic and replica swap attempts are probabilistic. In typical p-bit APT, the great majority of the random numbers are used for neuron updates rather than replica swap attempts. In contrast, c-bit APT requires orders of magnitude fewer random numbers. However, this potential benefit critically depends on the hardware cost of implementing c-bits, which may be greater than that of p-bits. Given the relative ease of creating pseudo random number generators through compact linear-feedback-shift-register circuits [67], c-bit implementations may in practice surreptitiously contain PRNG-like circuits. While p-bits and c-bits appear to be similar in their computational power, it is important to note that only p-bits enjoy a mathematical certainty that they sample from the Boltzmann distribution at steady state. These hardware implications are discussed further in the hardware discussion section of this chapter.

The algorithm is referred to as “adaptive” because it employs an instance-specific preprocessing method that finds a suitable β schedule for a given problem. This process ensures that replica-swap probabilities are nearly uniform, avoiding bottlenecks during the exchange process. The preprocessing details, including the practical procedure used to construct the β schedule, are deferred to Appendix A. The complete APT algorithm is summarized in pseudocode in Algorithm 1.

Algorithm 1: Adaptive Parallel Tempering with p-bits or c-bits

Input: weights J_{ij}, biases h_i, number of replicas N, inverse temperature profile β, time steps per swap, simulation time t_a

Output: Spin states $\{m\}$ corresponding to the minimum energy E_0

Initialize N system replicas to random spins

for time $t = 0$ to t_a **do**

  for N replicas at different β **do**

    Simulate p-bit Monte Carlo sweeps using Eq. 2.1 and 2.2, or c-bit dynamics using Eq. 2.2 and 2.5

  if time t is a multiple of time steps per swap **then**

    // Probabilistic swap attempt

    if it is an even numbered swap attempt **then**

      Select (even, odd) replica pairs

    else

      Select (odd, even) replica pairs

    for each replica **do**

      Compute energy $E\{m\}$ (Eq. 2.4)

    for each pair of adjacent replicas **do**

      if the Metropolis criterion is met (Eq. 2.17) **then**

        Swap the spin state $\{m\}$ between replicas

        // For c-bits, also swap the billiard state $\{x\}$ between replicas

2.5 Combinatorial Optimization Benchmarks

2.5.1 3D Spin-Glass Optimization

We consider 300 spin glass instances on a cubic lattice with side length $L = 7$ and $J_{ij} \in \{-1, +1\}$, borrowed from Ref. [60]. Purely probabilistic APT (p-bits) and hybrid chaotic-probabilistic APT (c-bits) are performed using similar parameters. We conduct APT pre-processing on 1 spin glass instance, returning a β schedule for 14 system replicas. In principle, to achieve the best performance, one could apply the APT pre-processing to each of 300 different problem instances and obtain 300 unique β schedules. Since this is a comparative study between p-bits and c-bits, we forego this step. Tuning parameters is not of particular concern.

For a given simulation time t_a, each of 300 spin glass instances is optimized using APT. For each simulation, we record the lowest energy E sampled across all replicas. We introduce residual energy as an optimization performance metric, defined as:

$$\rho_E^f = \frac{\langle E - E_0 \rangle}{n} \tag{2.18}$$

where E_0 is the ground state energy and n is the number of spins in the system. Fig. 2.5 shows ρ_E^f vs. computational time, where the time axis accounts for the total time steps summed across all $N = 14$ replicas that run in parallel. We take an ensemble average over 40 random seeds that randomize initial states for p-bits and c-bits, and phases for c-bits.

We devise a similar experiment for simulated annealing. In order to draw a comparison to APT, simulated annealing is conducted 14 times in parallel, with β increasing linearly from 0 to 10 during simulation time t_a. We record the lowest energy sampled across all replicas to compute the residual energy ρ_E^f. We take an ensemble average over 40 random seeds.

To compare p-bit and c-bit performance, we employ similar power-law scaling arguments as in the critical-scaling analysis earlier in this chapter. Specifically, $\rho_E^f = A t_a^{-\kappa_f}$, where A is a constant accounting for the difference in p-bit and c-bit prefactors, and κ_f is the power

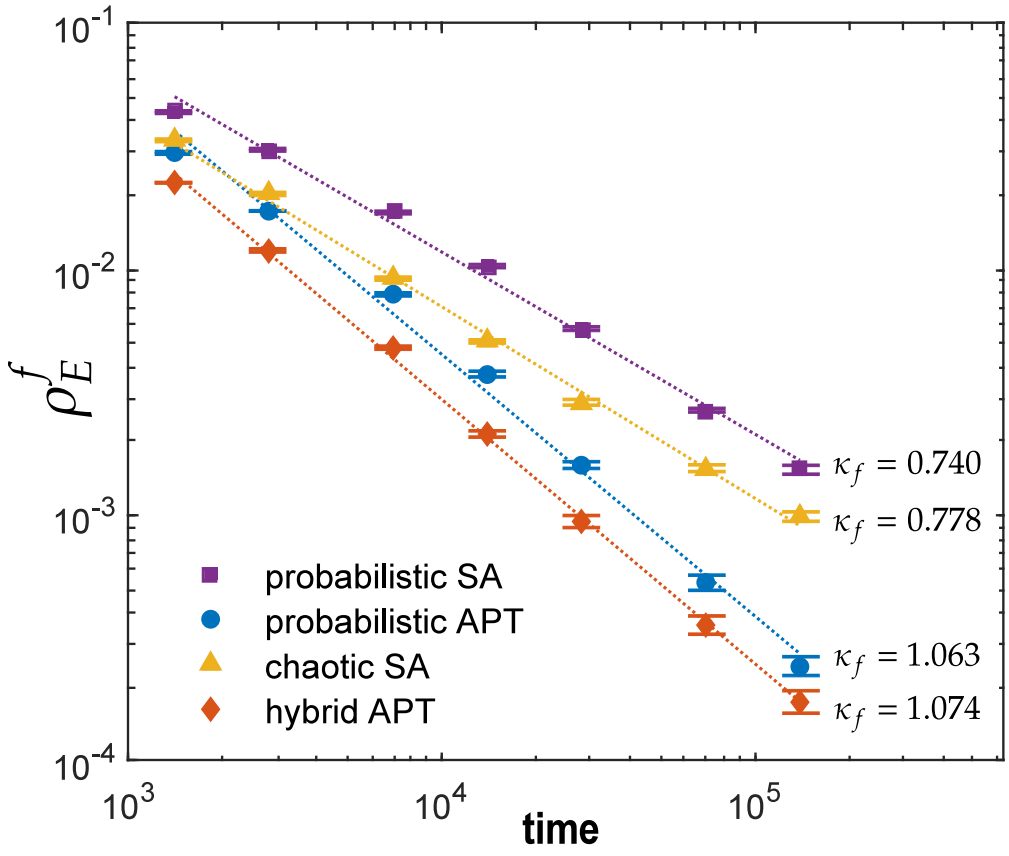


Figure 2.5: **Optimization of the 3D spin glass problem.** Residual energy density (ρ_E^f) as a function of total computational time (Nt_a) for cubic spin-glass instances with side length $L = 7$ ($J_{ij} \in \{-1, +1\}$). Curves scale as a power law: $\rho_E^f \propto t_a^{-\kappa_f}$. Time is measured in Monte Carlo sweeps for p-bits and time is dimensionless for c-bits. Error bars show 95% bootstrap confidence intervals. Results include p-bit simulated annealing (SA, purple squares), probabilistic adaptive parallel tempering (APT, blue circles), c-bit SA (yellow triangles), and c-bit APT with stochasticity (orange diamonds).

law scaling exponent. Fig. 2.5 shows that p-bit SA exhibits a similar scaling exponent κ_f to c-bit SA, while purely probabilistic APT yields a similar κ_f as hybrid probabilistic-chaotic APT. Furthermore, the APT algorithms show κ_f of larger magnitude than the SA algorithms, indicating a faster convergence to the ground energy E_0. Two conclusions can be drawn from these results. First, in the optimization setting, there is a strong similarity between c-bits and p-bits since they essentially show the same scaling for both SA and APT. Second, the hybrid APT algorithm proposed for c-bits shows superior performance over its fully deterministic counterpart.

Besides better algorithmic scaling, the APT algorithm proposed here enjoys another benefit: because coupling strengths are not adjusted globally, overflow or underflow issues related to the lifetime of c-bits become less of a concern, and they are a one-time problem to solve. For instance, past definitions of the c-bit have dx_i/dt that grow exponentially as β increases, so overflows may occur at cold temperatures [40].

The hybrid chaotic-probabilistic APT algorithm may be applicable to other approaches, such as the noise-injected oscillator-based Ising machines proposed in Ref. [33], that are designed to enable sampling from the Boltzmann distribution. However, the simplicity of the c-bit and its state-based, rather than phase-based, representation through explicit latches may still be more appealing.

2.5.2 Semiprime Factorization with Invertible Boolean Logic

We next consider a 10-bit × 10-bit invertible multiplier circuit composed of p-AND gates and p-Full Adders [2, 1] (Fig. 2.6a). This multiplier is represented by an undirected network of $n = 310$ p-bits with 1200 weighted connections, normalized such that $J_{ij} \in [-1, 1]$. Of the 310 spins in the system, 20 specific spins represent the product bits, while another set of 20 specific spins represents the two 10-bit factors. Using bipolar spin states ($m_i \in \{-1, +1\}$), traditional binary 0s are instead represented as negative spins $m_i = -1$. Clamping the multiplier's 20-bit output to a semiprime number configures the system such that its ground state, E_0, solves for

the correct prime factors. To accommodate semiprime numbers that are smaller than 20 bits, we pad the most significant bits with zeros. This scheme is referred to as invertible Boolean logic, in which p-bit based logic gates are run in reverse, similar in spirit to those used in memcomputing [2, 68].

We employ identical parameters for both probabilistic APT and hybrid chaotic-probabilistic APT. The APT pre-processing method finds a suitable β profile (Fig. 2.6c). We consider $N = 17$ system replicas, each running at a different inverse temperature β for a duration of $t_a = 10^5$ time steps. Probabilistic swap attempts are made between adjacent replica pairs once every 10 time steps, according to the Metropolis criterion (Equation 2.17). These parameters yield $Nt_a = 17 \times 10^5$ time steps of total computation summed across replicas.

During an APT simulation of $t_a = 10^5$ time steps, if the coldest replica (highest β) detects the correct factors at any time step, the simulation is considered a successful solve (Fig. 2.6b). Only the coldest replica is considered to remove the effect of random search statistics which may play a role in small problem sizes. For our comparative study, optimizing the algorithms' performance is not necessary.

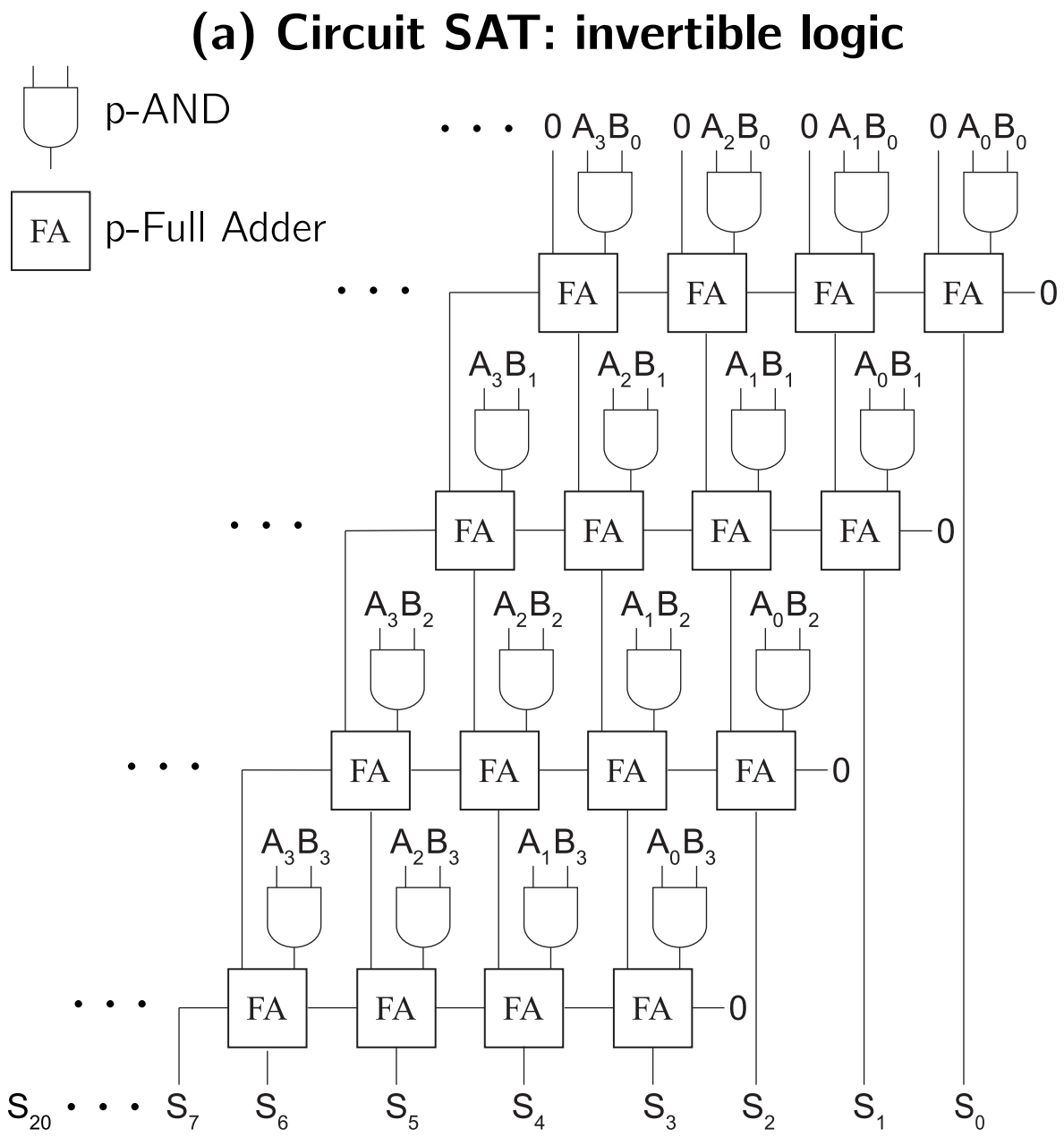


Figure 2.6: **Semiprime factorization with adaptive parallel tempering.** (a) p-bits are utilized to construct AND gates and full adders, that are the fundamental gates for a multiplier. The product bits are clamped to a semiprime number as in invertible Boolean logic [2].

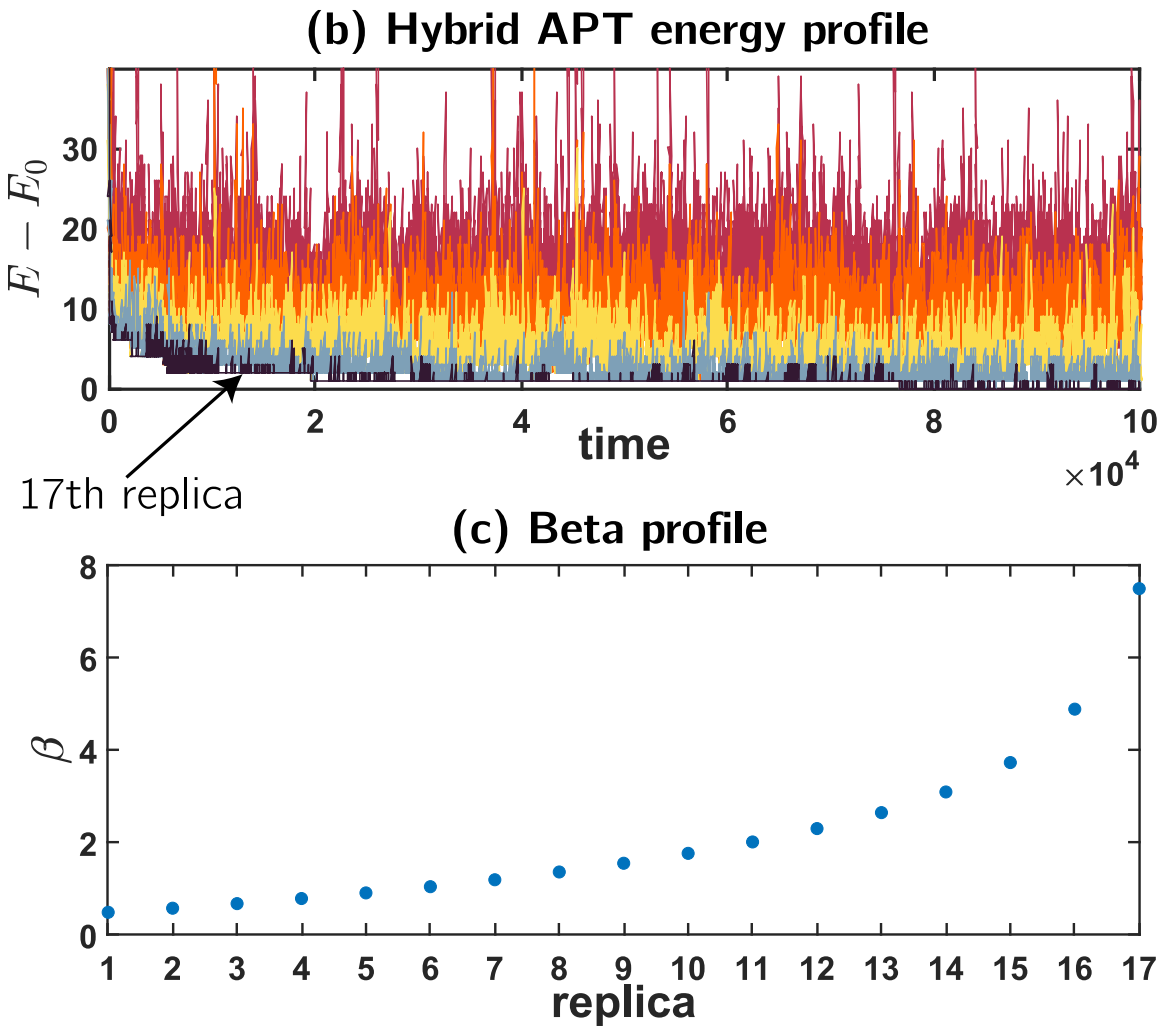


Figure 2.6: **Semiprime factorization with adaptive parallel tempering (continued).** (b) Energy profile of the 5 coldest replicas during hybrid chaotic-probabilistic adaptive parallel tempering (APT). $E - E_0$ represents distance from ground state energy. (c) β profile obtained from APT preprocessing.

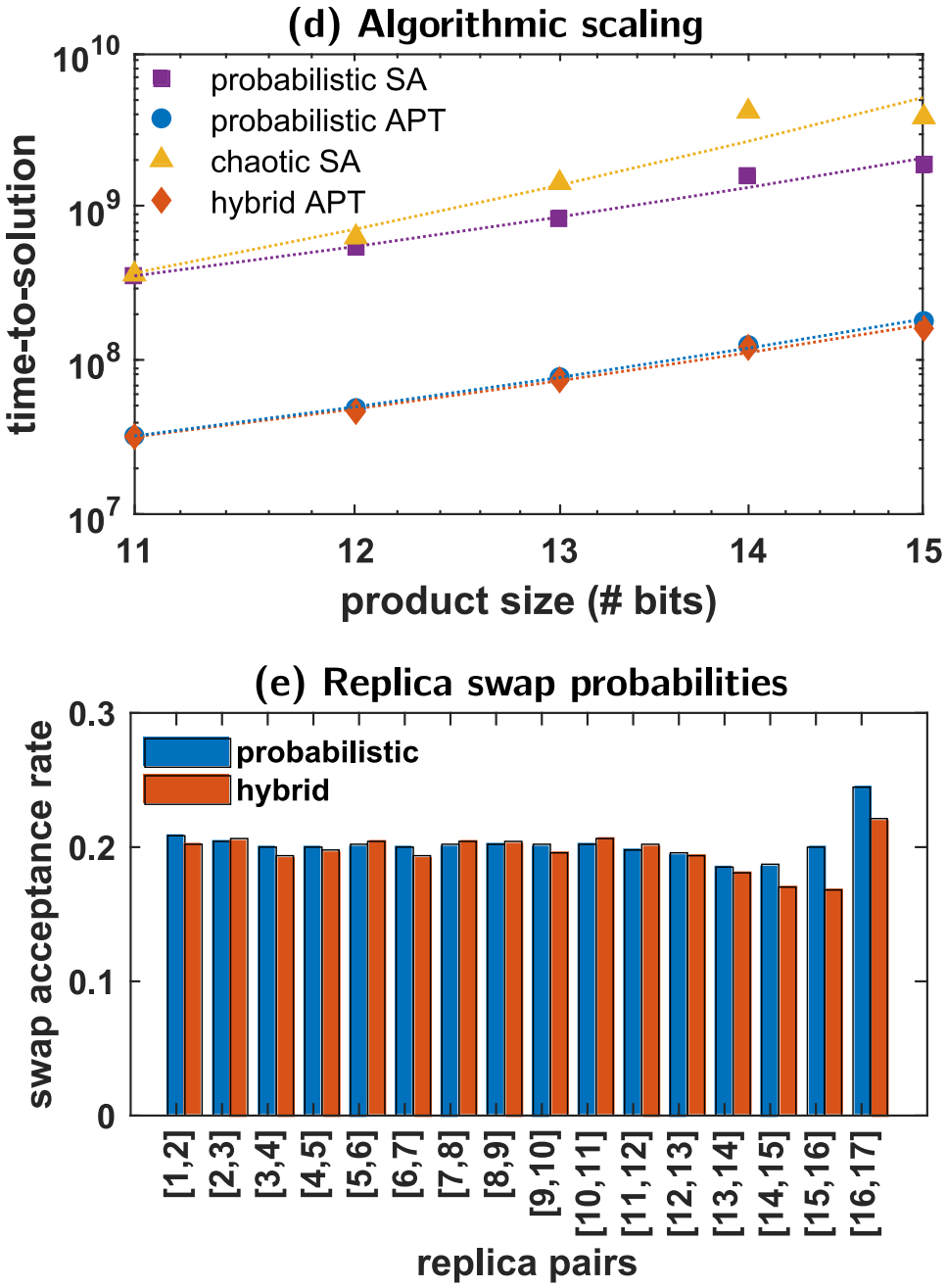


Figure 2.6: **Semiprime factorization with adaptive parallel tempering (continued).** (d) Time-to-solution vs. product size using a 10-bit × 10-bit factorizer for semiprime numbers. p-bit simulated annealing (SA, purple squares) yields a slope of 0.441, while p-bit APT (blue circles) yields a slope of 0.438. c-bit SA (yellow triangles) exhibits a slope of 0.658, and hybrid APT (orange diamonds) has a slope of 0.420. (e) Average replica swap probabilities from factoring a specific semiprime number over 100 trials (p-bit APT shown in blue, hybrid APT shown in orange).

We adopt a commonly used time-to-solution (TTS) metric [69]:

$$\langle \text{TTS} \rangle = (Nt_a) \left\langle \frac{\ln(1-0.99)}{\ln[1-p(Nt_a)]} \right\rangle. \tag{2.19}$$

Here, Nt_a is the total computational time summed across all replicas. The time-to-solution represents the amount of computation necessary to factor semiprimes with a 99% success probability. This expression follows by setting the probability of failing k independent attempts to $[1-p(Nt_a)]^k = 1-0.99$, solving for k, and multiplying the resulting number of attempts by the computational time Nt_a. For a given product size (number of bits), we consider 180 unique semiprime numbers. We attempt to factor each semiprime number 80 times. $p(Nt_a)$ is measured as the number of successful solves divided by the total number of attempted factorizations.

We devise a similar scheme for simulated annealing for the sake of comparison. We anneal 17 system replicas in parallel. Each replica runs for $t_a = 10^5$ time steps, and β is linearly increased from 0 to 10. In order to avoid random-find effects due to small problem sizes, we only check for the correct factors at the last 1000 time steps. If any of the 17 replicas finds the correct factors, the trial is considered a successful solve. For a given problem size (# bits), we again consider 180 unique semiprimes. We attempt to factor each semiprime 120 times. $p(Nt_a)$ and time-to-solution are calculated in a fashion analogous to APT.

To rule out implementation-dependent differences in unit time between c-bits and p-bits, we perform a scaling analysis, where TTS vs. problem size is assumed to have exponential dependence: $\text{TTS} = A\exp(kx)$, where x is the product size in bits, while A and k are constants.

Using this method of comparison, Fig. 2.6d shows that purely probabilistic APT exhibits similar algorithmic scaling to the proposed chaotic-probabilistic hybrid scheme. Moreover, they both exhibit better scaling than SA-based approaches. Pure chaotic SA, which has been commonly implemented [40], shows the worst scaling. Furthermore, both p-bits and c-bits observe a boundary effect as the problem size approaches 20 bits, where the time-to-solution actually decreases for all the algorithms. This is a feature particular to the fixed 20-bit factorizer

circuit: since numbers larger than 20 bits cannot be represented by the circuit, the solution space is reduced as we approach 20 bits. We exclude this data because it is uninformative for algorithmic scaling.

The two computing schemes not only share similar algorithmic scaling but also exhibit similarities in their underlying mechanisms. We consider a specific 11-bit semiprime number and conduct APT for $t_a = 10^5$ time steps. We average over 100 random seeds. Fig. 2.6e shows the probability of the Metropolis criterion dictating a swap between a given pair of replicas. The hybrid APT scheme follows a similar qualitative distribution as the probabilistic APT scheme.

The choice of integer factorization as a benchmark problem is simply due to the challenging combinatorial nature it poses to stochastic local search algorithms [70]. Specialized factoring algorithms starting with trial division and more complex field sieve approaches exhibit superior algorithmic scaling. It is important to note, however, that the factoring representation we use in this chapter is based on the more general NP-hard Circuit-SAT problem, for which no generic algorithmic speedup is known. As such, searching for scaling and prefactor improvements by dedicated hardware is highly desired.

2.6 Hardware Implications and Discussion

The potential hardware implementation of chaotic Ising machines without any explicit random number generators, while retaining similarity to the mathematics of p-bits, is appealing. Based on the original c-bit definition, which is different from the bounded-velocity definition used in this chapter, chaotic-bit implementations have been proposed in CMOS/analog VLSI and digital FPGA forms [48, 49, 71]. We now make some remarks regarding hardware implementations of chaotic networks; however, a concrete discussion of hardware implementations is beyond the scope of this chapter.

As a first evaluation of the algorithms considered in Sections 2.1–2.5, the results presented in this chapter are hardware-agnostic and based on scaling laws and algorithmic exponents.

However, there are restrictions that would be encountered in any hardware implementation.

One hardware-facing modification used earlier in this chapter is the bounded c-bit velocity. Most existing work employs a definition of c-bit billiard velocity (dx_i/dt) that grows exponentially at cold temperatures ($\beta \gg 1$), with one exception in prior work [45]. This is problematic because optimization algorithms such as simulated annealing and adaptive parallel tempering require cold temperatures, which may result in unbounded slopes. In this chapter, we employ a c-bit definition that has a maximum billiard speed of 2, as given in Eq. 2.5, making it more suitable for software and hardware implementation. This bounded c-bit choice is also a potential improvement over previous hardware implementations, which may have suffered from accuracy losses due to exponentially growing slopes. For example, the anomalous increase of energy plots in a prior FPGA implementation is likely due to overflows and underflows in slopes [47].

A second restriction is related to the synapse calculation time. In our software implementation, the synapse in Eq. 2.2 is updated every time step, so c-bit states are always updated using up-to-date information from their neighbors. Just as in asynchronous networks of p-bits [53], the asynchronous operation of c-bits requires a fast local-field calculation. This places stringent requirements on the network topology, since dense networks will require more time to update the synapse.

A third restriction is related to the latch mechanism. As discussed in Section 2.2, c-bit networks rely on sufficiently separated arrival times so that coupled latches do not update pathologically in the same effective time step. Hardware implementations will have analogous restrictions from finite setup/hold times and synapse delays, especially in dense network topologies.

Beyond digital implementations, c-bits can make use of “natural” latches. For example, the physics of spin-torque switching of a non-volatile nanomagnet [72] can function as the latch: the magnet will switch only if a current threshold is reached, and the magnet will exhibit hysteresis until the current threshold changes sign, much like the billiard-ball metaphor used in

c-bits.

A purported benefit of c-bits is the absence of explicit random numbers in their update rules. However, it is not clear whether the cost of c-bit hardware implementation is significantly cheaper than that of p-bits. First, low-cost pseudo-random number generators, such as LFSRs, do not significantly increase the complexity of implementing a p-bit. For example, an n-bit LFSR requires about $32 \times n$ transistors to implement [67]. In contrast, constructing a c-bit digitally may be significantly more complex than this cost due to the necessity of an oscillator with a tunable duty cycle. Additionally, there may be "hidden" sources of randomness, such as floating-point round-off errors in simulation or thermal noise in physical implementation.

Augmenting c-bits with explicit randomness in powerful tempering algorithms such as adaptive parallel tempering, as considered in the APT and combinatorial-optimization sections of this chapter, could offer further benefits. We believe that while the lack of random numbers may be an important c-bit advantage, comparisons must be made in concrete implementations in order to be meaningful, for example against the most energy-efficient and compact hardware solutions of p-bits using magnetic nanodevices [4]. We refrain from making preemptive assumptions about the scalability and efficiency of c-bit hardware, as specific implementation details are unknown. In addition, p-bit Monte Carlo sweeps are not in one-to-one correspondence with the c-bit time variable t, so speculation is not particularly informative until implementation. Nevertheless, the similar algorithmic scaling performance of c-bits and p-bits across the wide variety of problems considered in this chapter is very promising, providing valuable insights into the level of stochasticity required for effective probabilistic computing.

2.7 Chapter Summary

This chapter evaluated and improved deterministic chaotic bit (c-bit) networks that combine billiard dynamics with latches. While algorithm parameters were not excessively tuned, using similar parameters for probabilistic-bit (p-bit) and c-bit versions of simulated annealing (SA) and

adaptive parallel tempering (APT) enabled a fair comparison of algorithmic scaling performance. Augmenting deterministic c-bits with stochasticity through probabilistic Metropolis replica swaps produced a hybrid chaotic-probabilistic APT scheme with algorithmic scaling comparable to APT using fully stochastic networks of p-bits. This same hybrid scheme also achieved better algorithmic scaling than fully deterministic c-bits running simulated annealing. Beyond c-bits, the APT approach considered in this chapter is applicable to certain classes of oscillator Ising machines and similar nonlinear devices.

This chapter also showed that c-bits approximately sample from the quantum Boltzmann law in a one-dimensional transverse-field Ising model (TFIM). On two-dimensional Ising and three-dimensional spin-glass problems, c-bits qualitatively exhibit similar critical scaling dynamics as p-bits, although a precise correspondence between their measured scaling exponents κ_c remains unclear. While these critical dynamics are intriguing from a physics perspective, they are not essential to optimization, for which c-bits show strong promise. Combining stochastic p-bits and deterministic c-bits in asynchronous and massively parallel hardware implementations could create powerful domain-specific computers for combinatorial optimization and probabilistic sampling. In this chapter, "sampling" refers to trajectories through the exponentially large bitstring space of an Ising model. The transformer chapters apply the same sparse-sampling principle to value-cache rows in attention and, later, to FFN channel gates.

Chapter 3

Stochastic Sparse Attention for Memory-Bound Inference

3.1 Long-Context Autoregressive Decoding and the KV Bottleneck

This chapter moves the same principle to transformer inference, where the sampled object is a small set of value rows from a much larger attention context.

Transformers [41] underpin modern language models [42, 43, 44]. As deployments increasingly rely on long contexts, autoregressive decoding enters a regime where throughput is often limited by memory bandwidth: generating each new token repeatedly streams the Key–Value (KV) cache for all prior tokens. For example, for Llama-3.1-8B-Instruct at a 32k-token context, streaming bf16 keys and values is on the order of $\sim$128 MB *per layer per generated token*, and KV memory demands scale linearly with context length and batch size.

A broad literature mitigates this bottleneck from several angles. KV-cache quantization/compression reduces bytes per element [73, 74]; eviction policies reduce the number of retained tokens [75]; query-aware selection reduces the pages or tokens accessed per step while retaining the full cache [76]; and structured sparsity or candidate selection reduces attention

cost [77, 78, 79, 80, 81, 82]. Architectural changes such as grouped-query attention (GQA) reduce KV footprint by sharing keys/values across heads [83], and decoding kernels improve IO locality for *exact* attention [84, 85]. However, even with optimized exact kernels, long-context decoding still incurs a persistent cost: each step touches essentially the full KV state. In contrast to methods that decide which tokens or cache blocks are retained, evicted, or reconstructed, the goal of SANTA is to reduce the cost of the value aggregation operator over whatever KV state is made available at decode time.

Exact attention kernels and memory-bound decoding. FlashAttention-style kernels optimize IO locality and utilization for *exact* attention, including decoding-focused variants [84, 85]. These kernels are crucial baselines for long-context inference, but exact decoding still streams essentially the full KV state each step at long contexts. SANTA is complementary to this line of work: rather than further optimizing a full KV scan, it reduces the amount of KV data accessed at runtime by sparsifying value-cache reads. The forward-looking Bernoulli qK^{T} mechanism discussed later in this dissertation similarly targets key-feature reads, but the main algorithmic focus of this chapter is the softmax–V value aggregation.

Reducing KV footprint, retention, and selection. KV quantization/compression reduces bytes per KV element [73, 74], weight-only quantization targets model parameters [86, 87], and GQA reduces KV size through head sharing [83]. Recent long-context systems also reduce the effective cache through selection, retention, or offloading: ShadowKV [88], OmniKV [89], and DuoAttention [90] decide which KV states are retained, reconstructed, selected, or assigned to retrieval/streaming heads. These methods answer a different systems question from SANTA. They reduce or restructure the candidate KV state before attention is evaluated, whereas SANTA modifies the softmax–V aggregation over whatever KV state is presented to the operator. Thus, after an upstream method reduces the candidate set from n to n', SANTA can in principle replace dense aggregation over n' value rows with sampled aggregation over S rows. The marginal benefit will depend on n', hardware, batching, and task, so such hybrids are best viewed as complementary. Multi-Head Latent Attention (MLA)-style architectures [91] may

similarly reduce the KV bandwidth share targeted by SANTA, making them an important future benchmark.

Approximating attention and sampling vs. truncation. Structured sparsity and low-rank approximations reduce attention cost, often in prefill [77, 78, 79, 80]. In decoding, candidate selection and query-pruning methods reduce the cost of identifying relevant tokens or approximating $qK^\top$ [81, 82, 92]. SANTA operates in the value stage and is orthogonal to score-stage approximations, while Bernoulli $qK^\top$ is an alternative score-stage estimator that induces feature-sparse key access via stochastic ternary queries. Sampling-based attention approximations have also been explored [93]. A direct deterministic baseline is top-k attention [94], which reduces value reads by truncation but is biased. In contrast, SANTA provides an unbiased estimator of the post-softmax value aggregation with controllable variance, and the following sections emphasize variance reduction through stratified and systematic sampling. This sampling-based approach is not claimed to dominate deterministic truncation in all regimes; rather, the relevant comparison is regime-dependent and becomes especially important in the long-context, memory-bound decoding setting targeted here.

Contributions and scope. Concretely, the SANTA work adapted in this dissertation: (i) introduces SANTA and variance-reduced S^2ANTA as unbiased estimators of the post-softmax value aggregation that reduce value-cache reads from n_k to S; (ii) develops the unbiasedness, variance, concentration, and variance-reduction arguments that justify the estimator; (iii) empirically characterizes tunable accuracy–efficiency trade-offs on GSM8K, MMLU, and long-context benchmarks; and (iv) designs GPU kernels that parallelize sampling and sparse accumulation, demonstrating up to $1.5\times$ decode-step attention-kernel speedup and up to $1.25\times$ end-to-end decode-latency speedup in batched long-context generation while matching baseline long-context accuracy. This chapter develops the algorithmic and statistical structure of SANTA; the GPU-kernel realization and systems-level latency evaluation are deferred to the following chapter. A secondary Bernoulli $qK^\top$ score-stage extension is discussed in Chapter 4, but the main SANTA contribution and measured speedups come from value-stage sparsification.

3.2 Stochastic Additive No-mulT Attention

We introduce **S**tochastic **A**dditive **N**o-mul**T** **A**ttention (**SANTA**), which samples $S \ll n_k$ indices from the post-softmax distribution and aggregates only those sampled rows of V via gather-and-add. This yields an *unbiased estimator of the post-softmax value aggregation* while reducing value-cache row reads per query from n_k to S and replacing the dense value-stage weighted multiply-accumulate over all value rows with sampled gather-and-add, followed by the fixed normalization by $1/S$. We further propose variance-reduced variants (**S^{2}ANTA**) based on stratified and systematic sampling that bolster accuracy and admit GPU-friendly implementations. Since sparsifying V reads yields the clearest bandwidth benefit in decoding, the systems evaluation targets decode-step acceleration at long contexts.

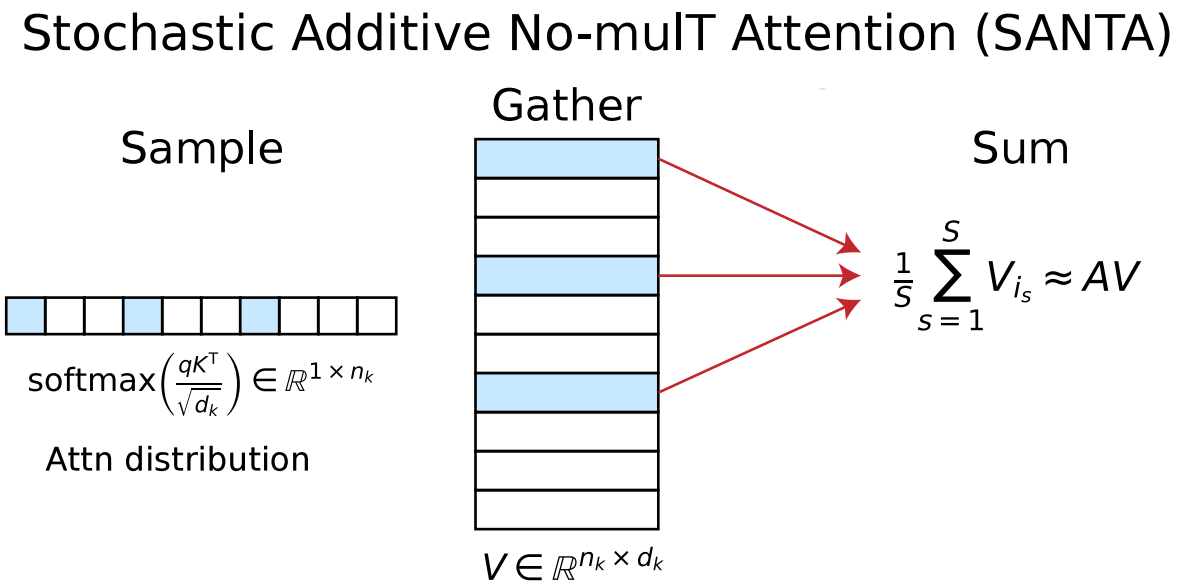


Figure 3.1: SANTA requires $S \ll n_k$ memory accesses to sampled rows of V, replacing the dense post-softmax weighted value sum with sparse gather-and-add. The remaining normalization by $1/S$ is a bit shift if S is a power of 2.

In principle, SANTA can be employed for the prefill or decode stages. Since the benefit of sparse memory access becomes apparent in decode, we present SANTA as a decoding algorithm. In autoregressive decoding, we consider a single query vector $q \in \mathbb{R}^{1 \times d_k}$. Scaled dot-product

attention (SDPA) is expressed as a function of the key matrix and value matrix:

$$\text{Attn}(q, K, V) = \text{softmax}\left(\frac{qK^{\top}}{\sqrt{d_k}}\right)V = AV, \tag{3.1}$$

where $K \in \mathbb{R}^{n_k \times d_k}$ is the key matrix, $V \in \mathbb{R}^{n_k \times d_k}$ is the value matrix, and $A \in \mathbb{R}^{1 \times n_k}$ represents the attention scores as a probability distribution.

To approximate AV, we treat A as a categorical distribution over keys and sample S indices i.i.d. from it with replacement. For query vector q, we fetch the corresponding S rows of V and average them. This yields an estimate of the attention output using only row indexing and addition.

As an illustrative example, consider $n_k = 3$ keys. We sample $S = 3$ one-hot attention vectors from the categorical distribution A:

$$\widehat{AV} = \frac{1}{3}\left(\begin{bmatrix}1 & 0 & 0\end{bmatrix} + \begin{bmatrix}1 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0 & 1\end{bmatrix}\right)V, \tag{3.2}$$

which simplifies by distributivity of matrix multiplication over addition:

$$\begin{aligned}\widehat{AV} &= \frac{1}{3}\left(\begin{bmatrix}1 & 0 & 0\end{bmatrix}V + \begin{bmatrix}1 & 0 & 0\end{bmatrix}V + \begin{bmatrix}0 & 0 & 1\end{bmatrix}V\right) \\ &= \frac{1}{3}(V_1 + V_1 + V_3).\end{aligned} \tag{3.3}$$

Here, each $V_i \in \mathbb{R}^{1 \times d_k}$ is a sampled row of V. The one-hot attention vectors in Eqs. 3.2, 3.3 are only for illustrative purposes; they represent an indexing and gather operation, and are not materialized as vectors in memory.

Finally, SANTA can be summarized as:

$$\widehat{AV} = \frac{1}{S}\sum_{s=1}^{S} V_{i_s} \approx AV, \tag{3.4}$$

where i_s is a sampled index and each V_{i_s} denotes a gathered row from V. The dense value-stage weighted multiplications have therefore been removed from the part of the computation

that scales with n_k; SANTA uses sampling, indexing, and addition, followed by a fixed d_k-dimensional normalization by $1/S$. If S is chosen as a power of 2 and the model has a fixed point representation, normalization may be implemented as a bit-shift. Fig. 3.1 illustrates SANTA's memory access benefits for autoregressive decoding: SANTA sparsely reads only $S \ll n_k$ sampled rows of V and computes a sampled additive average.

The formal treatment of unbiasedness, variance scaling, and concentration for this estimator is given in Section 3.3.

3.3 Unbiasedness, Variance, and Concentration

This section gives the main probabilistic guarantees for SANTA. The key point is that SANTA replaces the dense value-stage weighted sum by a Monte Carlo average over value rows sampled from the attention distribution. The resulting estimator is unbiased for the usual attention output, its covariance decreases as $1/S$, and a vector Bernstein inequality gives a high-probability concentration bound. Complete derivations and proof details are deferred to Appendix A.2.1–A.2.4.

Let

$$A = \mathrm{softmax}\left(QK^{\top}/\sqrt{d_k}\right) \in \mathbb{R}^{n_q \times n_k}, \tag{3.5}$$

where $Q \in \mathbb{R}^{n_q \times d_k}$, $K \in \mathbb{R}^{n_k \times d_k}$, and $V \in \mathbb{R}^{n_k \times d_k}$. For each query index $q \in \{1, \dots, n_q\}$, SANTA draws

$$i_{q,1}, \dots, i_{q,S} \overset{\text{i.i.d.}}{\sim} \mathrm{Categorical}(A_q),$$

and returns

$$\widehat{AV}_q = \frac{1}{S}\sum_{s=1}^{S} V_{i_{q,s}}, \qquad i_{q,s} \sim \mathrm{Categorical}(A_q). \tag{3.6}$$

Stacking these rows gives the estimator $\widehat{AV} \in \mathbb{R}^{n_q \times d_k}$. Thus, for each query, SANTA forms an additive average of S gathered value rows rather than explicitly computing the full weighted sum over all n_k value rows.

Unbiasedness. For a fixed query q, define the exact attention output

$$\mu_q \triangleq \sum_{j=1}^{n_k} A_{qj} V_j = (AV)_q.$$

Since $\Pr(i_{q,s} = j) = A_{qj}$, each sampled value row has expectation μ_q. Therefore,

$$\mathbb{E}[\widehat{AV}_q] = \frac{1}{S} \sum_{s=1}^{S} \sum_{j=1}^{n_k} A_{qj} V_j = \sum_{j=1}^{n_k} A_{qj} V_j = (AV)_q. \tag{3.7}$$

Equivalently, after stacking all query rows,

$$\mathbb{E}[\widehat{AV}] = AV. \tag{3.8}$$

Thus, SANTA is an unbiased estimator of the standard attention value output. The full statement and proof are given in Appendix A.2.2.

Variance scaling. The variance of SANTA decreases linearly with the number of samples. Fix a query q, write $p_{qj} \triangleq A_{qj}$, and define the query-specific covariance

$$\Sigma_q \triangleq \mathrm{Cov}(V_i) = \sum_{j=1}^{n_k} p_{qj} (V_j - \mu_q)(V_j - \mu_q)^\top, \tag{3.9}$$

where $i \sim \mathrm{Categorical}(A_q)$. Then

$$\mathbb{E}[\widehat{AV}_q] = \mu_q, \qquad \mathrm{Cov}(\widehat{AV}_q) = \frac{1}{S} \Sigma_q. \tag{3.10}$$

Equivalently, the mean-squared error satisfies

$$\mathbb{E}\left[\|\widehat{AV}_q - \mu_q\|_2^2\right] = \frac{1}{S} \mathrm{tr}(\Sigma_q). \tag{3.11}$$

This is the standard Monte Carlo variance scaling: increasing the sample count S reduces

the covariance and mean-squared error by a factor of $1/S$. The full derivation is given in Appendix A.2.2.

Concentration. The same estimator also admits a dimension-free high-probability tail bound. Assume a uniform almost-sure bound

$$\|V_j - \mu_q\|_2 \le V_{\max}$$

for all j, and define

$$\nu_q \triangleq \mathbb{E}\left[\|V_i - \mu_q\|_2^2\right] = \operatorname{tr}(\Sigma_q).$$

Then, for all $t > 0$,

$$\Pr\left(\left\|\widehat{AV}_q - \mu_q\right\|_2 \ge t\right) \le 2\exp\left(-\frac{S\,t^2}{2\,(\nu_q + V_{\max} t/3)}\right). \tag{3.12}$$

Equivalently, with probability at least $1 - \delta$,

$$\left\|\widehat{AV}_q - \mu_q\right\|_2 \le \sqrt{\frac{2\,\nu_q\,\log(2/\delta)}{S}} + \frac{2V_{\max}\log(2/\delta)}{3S}. \tag{3.13}$$

This bound follows from a Hilbert-space Bernstein inequality for independent mean-zero bounded vectors [95, Thm. 3.4]; the specialization to SANTA is detailed in Appendix A.2.4. The leading term scales as $S^{-1/2}$ in norm, while the corresponding covariance and mean-squared error scale as $1/S$.

3.4 Variance-Reduced Sampling

The iid SANTA estimator reduces the value-stage aggregation to sampled row gathers and additions, but independent sampling can repeatedly draw the same high-probability keys and leave other portions of the attention distribution under-covered. S^2ANTA reduces this variance

by stratified sampling: dividing the attention CDF into S equal probability mass intervals and sampling once per interval. This dissertation uses two variants: independent stratified sampling (S^2**ANTA-strat**) and systematic sampling (S^2**ANTA-sys**).

The mechanics of S^2ANTA-strat remain identical to default SANTA, except that one sample is drawn from each of the S strata instead of drawing S i.i.d. samples from the full distribution. Systematic sampling similarly splits the attention distribution CDF into S equal probability mass partitions. However, whereas independent stratified sampling draws one independent random offset for each of S strata, systematic sampling draws a single offset U and applies the same offset to all strata, effectively drawing S samples with one random number.

Construction. Fix a query q. Let $A_q = (p_{q1}, \dots, p_{qn_k})$ be the attention weights and F_q the associated CDF on $[0, 1)$. Partition $[0, 1)$ into S equal intervals

$$I_m := [m/S, (m+1)/S), \qquad m = 0, \dots, S-1.$$

Define two schemes:

- **Independent stratified (S^2ANTA-strat).** Draw $T_m \sim \text{Unif}(I_m)$ independently for $m = 0, \dots, S-1$, set $J_m := F_q^{-1}(T_m)$, and output

$$\widehat{AV}_q^{\text{ind}} := \frac{1}{S} \sum_{m=0}^{S-1} V_{J_m}.$$

- **Systematic (S^2ANTA-sys).** Draw a single $U \sim \text{Unif}([0, 1/S))$ and take thresholds $T_m := U + m/S$. With $J_m := F_q^{-1}(T_m)$, output

$$\widehat{AV}_q^{\text{sys}} := \frac{1}{S} \sum_{m=0}^{S-1} V_{J_m}.$$

Both S^2ANTA-strat and S^2ANTA-sys remain unbiased estimators of the dense attention

output. Let

$$\mu_q := \sum_j p_{qj} V_j = (AV)_q.$$

Then

$$\mathbb{E}[\widehat{AV}_q^{\text{ind}}] = \mathbb{E}[\widehat{AV}_q^{\text{sys}}] = \mu_q.$$

For S^2ANTA-strat, this follows by averaging one uniform draw over each equal-mass interval. For S^2ANTA-sys, each threshold T_m is still marginally uniform on its stratum, although the thresholds are dependent across strata; this dependence changes the variance behavior but not the expectation. The formal unbiasedness argument is given in Appendix A.2.3, following standard treatments of stratified and systematic sampling [96, 97].

Only S^2ANTA-strat has a general variance-reduction guarantee relative to default iid SANTA. Let

$$\Sigma_q := \sum_j p_{qj} (V_j - \mu_q)(V_j - \mu_q)^\top$$

be the value covariance under the attention distribution, and define the within-stratum covariances

$$\Sigma_q^{(m)} := \mathrm{Cov}\Big(V_{F_q^{-1}(T)} \,\Big|\, T \sim \mathrm{Unif}(I_m) \Big).$$

Then independent stratification gives

$$\mathrm{Cov}\big(\widehat{AV}_q^{\text{ind}}\big) = \frac{1}{S^2} \sum_{m=0}^{S-1} \Sigma_q^{(m)} \;\preceq_{\text{Loewner}}\; \frac{1}{S}\,\Sigma_q = \mathrm{Cov}(\widehat{AV}_q),$$

with strict improvement unless all stratum means coincide. Equivalently,

$$\mathbb{E}\Big[\|\widehat{AV}_q^{\text{ind}} - \mu_q\|_2^2\Big] \le \mathbb{E}\big[\|\widehat{AV}_q - \mu_q\|_2^2\big] = \frac{1}{S}\,\mathrm{tr}(\Sigma_q).$$

This variance-dominance statement follows from the law of total covariance and is detailed in Appendix A.2.3 [98, 96].

The corresponding systematic estimator is also unbiased, but it does not provide a tractable

general variance guarantee. Its advantage is implementation simplicity: S^2ANTA-sys uses one random number per query instead of S random numbers. Despite lacking the same formal variance-reduction theorem, S^2ANTA-sys is validated empirically and exhibits variance behavior comparable to S^2ANTA-strat in practice.

To validate the variance behavior directly, this chapter measures the empirical variance of SANTA, S^2ANTA-strat, and S^2ANTA-sys by sweeping the sample budget S on 8k-token single-hop QA prompts, averaged across model layers and 30 prompts and normalized per head. The S^2ANTA variants exhibit significantly lower variance than default SANTA. On a log-log scale, the slopes for SANTA, S^2ANTA-strat, and S^2ANTA-sys are -1.1, -1.27, and -1.29, respectively, matching the $1/S$ scaling prescribed by theory. The measurement protocol and empirical variance plot are deferred to Appendix A.2.6.

A related empirical quantity is the number of unique keys sampled. Since all versions of SANTA sample keys with replacement,

$$U := \left|\{J_1, \dots, J_S\}\right|$$

can be smaller than the nominal sample budget S. In 8192-token single-hop QA prompts, the average unique-key counts are

S	32	64	128	256
SANTA	11.0	16.7	25.9	38.3
S^2ANTA-strat	12.7	21.2	35.8	60.3
S^2ANTA-sys	13.0	21.7	36.4	61.6

Thus, for a fixed budget, the variance-reduced samplers provide higher key diversity than default SANTA, while the number of unique rows of V remains well below S. Details of the aggregation procedure and unique-key measurements are given in Appendix A.2.6.

3.5 Theoretical Compute and Memory Accesses

This section separates the theoretical savings from SANTA in autoregressive decoding from the corresponding accounting in prefill. SANTA changes the post-softmax value aggregation stage, where the attention scores A multiply the value matrix V. In the analysis below, the score computation $qK^\top$ is treated as similar for SANTA and dense scaled dot-product attention (SDPA), so the clearest savings come from the value stage rather than from avoiding the score-stage key access.

Autoregressive decoding. In autoregressive decoding, each generated token corresponds to a single query attending to n_k cached keys and values. By virtue of SANTA's sparsity, SANTA reduces the post-softmax additions and reads to rows of V to an S/n_k fraction of the dense SDPA value-stage cost, where S is the sample budget and n_k is the number of keys. SANTA also removes the dense post-softmax multiplications in favor of additive accumulation, leaving only a d_k-dimensional normalization.

Table 3.1: Per-query, per-head value-stage compute and memory for dense scaled dot-product attention (SDPA) and SANTA. n_k is the number of keys, d_k is the head dimension, and S is the SANTA sample budget per query. Value reads count V elements accessed per query; writes count output elements.

Metric	SDPA	SANTA
Adds	$n_k d_k$	$S d_k$
Mults / Divs	$n_k d_k$	d_k
Value reads	$n_k d_k$	$S d_k$
Writes	d_k	d_k

Table 3.1 displays the theoretical additions, multiplications, and memory accesses incurred by SANTA in the value stage of attention. We count individual scalar elements which are added together, multiplied together, or accessed by memory. Architecture-dependent effects such as multiply-accumulate (MAC) instructions, caching effects, or operation parallelism are not accounted for; the arguments are fundamental and hardware-agnostic.

A key distinction is that SANTA costs *fewer* additions than SDPA; SANTA does not seek to replace the multiplications in the AV computation by another dense weighted sum, but

rather leverages sparse sampling to decrease the total number of additions while eliminating the dense value-stage multiplications. So long as $S \ll n_k$, the advantage of attention sparsity holds true. Sampling overhead is not included in this theoretical accounting because it is implementation-dependent. Whereas SDPA incurs $n_k d_k$ multiplications from computing the dense weighted value sum, SANTA only costs d_k normalizing multiplies by $1/S$. Since d_k is a fixed model parameter, this normalization does not increase with the sequence length n_k. Furthermore, if S is chosen as a power of 2 and the model uses a fixed-point representation, this normalization may be implemented as a bit-shift.

Prefill. In prefill, similar arguments hold for SANTA's reduced additions and removed dense value-stage multiplications on a per-query basis, but there is no longer an equally clear sparse memory-access benefit to the value matrix V when all queries are considered together. Each of n_q queries may sample S rows of V, therefore requiring $\gg S$ rows of V when the union across all queries is considered. In the common causal prefill setting, $n_q = n_k$, so sparse value access per query does not necessarily imply sparse unique value-row access over the entire prefill pass.

Table 3.2: Unique V rows read, measured as the union over all query positions in a causal prefill pass and reported as a percentage of all n_k rows in the value matrix. Each entry is the mean over 10 trials and all attention heads, with Gaussian inputs $Q, K, V \sim \mathcal{N}(0, 1)$ (FP16), H=32 heads, and head dimension d_k=128.

S	**Unique V rows read** (% of n_k)				
	n_k=1024	2048	4096	8192	16384
1	49.90	50.04	49.96	49.95	49.95
4	79.95	80.00	79.96	79.96	79.95
8	88.92	88.83	88.86	88.86	88.85
16	94.07	94.11	94.10	94.09	94.08

Table 3.2 shows the percentage of unique V rows read by SANTA during prefill passes of differing sequence lengths n_k. Even in the long-context, $S \ll n_k$ regime, any appreciable number of samples S results in SANTA sampling most rows of V when the union across all queries is considered. For $S = 16$, selected sequence lengths from n_k = 1k–16k show SANTA reading almost all ($> 94\%$) rows of V, because each of $n_q = n_k$ queries stochastically attends

to S sampled keys. Therefore, while SANTA features sparse V access during autoregressive decoding, SANTA does *not* have an apparent benefit from sparse unique V-row access in prefill. Arguments concerning SANTA's reduced addition and multiplication operations are still valid in prefill.

Table 3.3: Per-head FLOPs and memory for scaled dot-product attention (SDPA), top-k, and SANTA. n_q: number of queries, n_k: number of keys, d_k: head dimension, k: top-k budget, S: SANTA sample budget per query. "Reads" count logical value/key elements accessed; "Writes" count output or score elements. Softmax, top-k selection, and sampling overheads are omitted.

Stage / Variant	**Adds**	**Mults / Divs**	**Reads**	**Writes**
Score stage ($QK^\top$)				
SDPA / top-k / SANTA	$n_q n_k d_k$	$n_q n_k d_k$	$(n_q + n_k)d_k$	$n_q n_k$
Value stage				
SDPA	$n_q n_k d_k$	$n_q n_k d_k$	$n_q n_k d_k$	$n_q d_k$
Top-k	$n_q k d_k$	$n_q k d_k$	$n_q k d_k$	$n_q d_k$
SANTA	$n_q S d_k$	$n_q d_k$	$n_q S d_k$	$n_q d_k$

Table 3.3 summarizes the theoretical costs per attention head for the generalized prefill setting. For the score stage ($QK^\top$), the reads entry $(n_q + n_k)d_k$ counts each query and key vector once per head, assuming they are loaded once from DRAM and then reused in on-chip memory during the matrix multiply. In contrast, for the value stage we report $n_q n_k d_k$, $n_q k d_k$, and $n_q S d_k$ for dense SDPA, top-k, and SANTA, respectively, which count the per-query accesses rather than unique loads of elements from memory.

While top-k attention is memory-efficient and generally performs well, it introduces bias and can degrade significantly when the attention distribution is heavy-tailed. For example, recent work [82] shows that tasks such as common word extraction suffer when relevant information is not concentrated in the top-k entries. In contrast, stochastic estimators like SANTA provide an unbiased estimate of full attention and retain robustness in such cases.

For both top-k and SANTA, we define a notion of compute budget, where k is the number of keys selected by top-k for each query of each attention head, while S is a sample budget representing the number of samples that SANTA draws for each query of each attention head. This section assumes a uniform S across model layers. We omit softmax computation, top-k

sorting overhead, and sampling from this analysis, as they are lightweight relative to memory accesses and the V matrix multiply, and are highly implementation-dependent. For example, many implementations use partial sorting [99, 100].

In both FLOPs and memory reads, SDPA scales as $n_q n_k$, reflecting the cost of full quadratic attention computation. Both top-k and SANTA reduce this to $n_q k$ and $n_q S$, respectively, assuming fixed budgets with $k, S \ll n_k$. This shifts the overall complexity of the value stage from quadratic to linear in sequence length, given a fixed budget k or S. Top-k still incurs multiplication costs while SANTA removes the dense value-stage multiplications: top-k performs $n_q k d_k$ multiplications per head, while SANTA requires only $n_q d_k$ divisions or normalizing multiplies. If S is chosen as a power of 2, even these few divisions become simple bit-shifts in a fixed-point representation.

3.6 Accuracy on Reasoning and Long-Context Benchmarks

After establishing the SANTA estimator, its variance-reduced variants, and the theoretical compute and memory-access regimes in the preceding sections, we additionally test SANTA, S^2ANTA-strat, and S^2ANTA-sys as mathematical constructions on GSM8K, MMLU, and 8k-token long-context tasks using PyTorch reference implementations. These experiments apply SANTA in both prefill and decoding, so we use these results primarily as accuracy checks rather than speedup claims; sparse value-cache memory savings are most meaningful during autoregressive decoding. Additional DeepSeek-R1-Distill-Qwen-7B and top-k comparisons, prompting/parsing/grading details, and HumanEval code-generation checks are reported in Appendix Sections A.2.7, A.2.11, and A.2.16, respectively. The matched-budget top-k comparisons use $k = S$ as an accuracy comparison rather than an iso-compute benchmark, because SANTA removes the dense value-stage weighted multiplications apart from the fixed normalization counted in Table 3.1.

3.6.1 GSM8K

Table 3.4: GSM8K accuracy and average context length (prompt + answer) for Llama-3.1-8B-Instruct. S: SANTA sample budget. Accuracy shows 95% bootstrap confidence intervals.

	S^2ANTA-sys		S^2ANTA-strat		SANTA	
S	Acc. (%)	Tok.	Acc. (%)	Tok.	Acc. (%)	Tok.
2	1.11±0.34	1071	1.01±0.32	1071	1.26±0.34	1075
4	1.67±0.40	569	1.54±0.38	611	1.57±0.37	825
8	2.10±0.45	340	1.74±0.40	325	1.44±0.38	207
16	44.63±1.58	349	39.12±1.50	352	5.51±0.72	350
32	68.59±1.42	339	67.00±1.48	343	38.26±1.43	348
64	76.42±1.34	341	74.43±1.31	343	63.63±1.49	346
128	**77.33±1.34**	342	75.64±1.33	341	70.23±1.42	341
256	**77.56±1.34**	343	**78.17±1.28**	343	75.61±1.42	342
SDPA (baseline)					**78.06±1.33**	344

We consider Llama-3.1-8B-Instruct ("Llama 8B") [101] on the GSM8K dataset as a mathematical reasoning benchmark [102, 103]. Table 3.4 provides both accuracy and the average number of tokens (prompt + answer). Because the compute and memory access costs of SANTA are contextualized relative to the sequence length n_k, these sequence lengths justify the regimes where $S \ll n_k$ and SANTA is, in principle, cheaper than SDPA in attention's value stage.

Default SANTA exhibits respectable performance, but variance-reduced S^2ANTA variants demonstrate superior performance across every sample budget S. S^2ANTA implementations approach the accuracy of full SDPA (within 1%) while S remains shorter than the sequence length n_k, translating to memory-access savings and FLOP energy savings. In deployment settings where LLM users do not require maximum model performance, the sample budget S can be tuned to save computation cost. For instance, S^2ANTA-sys achieves 76.42% accuracy at $S = 64$, which is about 19% of the average sequence length of $n_k = 341$ tokens. For a modest accuracy trade-off compared to SDPA's 78.06%, S^2ANTA-sys in principle costs 19% of the value-stage additions and 19% of the V-matrix accesses, while avoiding the dense value-stage weighted multiplications apart from the fixed normalization counted in Table 3.1.

We provide additional DeepSeek-R1-Distill-Qwen-7B [104] (DeepSeek 7B) results and comparisons to sparse top-k attention in Appendix Section A.2.7.

3.6.2 MMLU

Table 3.5: MMLU accuracy and average context length (prompt + answer) for Llama-3.1-8B-Instruct. S: SANTA sample budget. Accuracy shows 95% bootstrap confidence intervals.

	S^2ANTA-sys		S^2ANTA-strat		SANTA	
S	Acc. (%)	Tok.	Acc. (%)	Tok.	Acc. (%)	Tok.
2	24.52±1.22	1141	24.28±1.21	1141	24.89±1.27	1144
4	24.49±1.18	774	24.65±1.26	807	25.13±1.20	961
8	24.47±1.23	292	25.63±1.23	286	25.06±1.23	284
16	34.14±1.39	353	32.46±1.32	350	26.24±1.26	313
32	45.48±1.45	403	43.78±1.50	398	33.81±1.34	354
64	**48.70±1.39**	418	**49.25±1.48**	415	41.11±1.40	397
128	**50.60±1.37**	421	**49.92±1.47**	421	47.46±1.45	412
256	**51.12±1.46**	421	**50.90±1.49**	422	**49.75±1.51**	417
SDPA (baseline)					**49.86±1.47**	424

We further evaluate SANTA on the MMLU benchmark [105, 106], which tests factual recall and general reasoning. MMLU benchmarking observes similar trends as GSM8K, where SANTA virtually recovers baseline accuracy at $S = 256$, which is significantly shorter than the average sequence length $n_k = 417$. S^2ANTA variants outperform default SANTA across S budgets of 64–256, recovering baseline SDPA accuracy (within 1%) while avoiding dense value-stage weighted multiplications and sparsely accessing rows of V since $S \ll n_k$. Extended results with DeepSeek 7B and top-k comparisons are provided in Appendix Section A.2.7.

3.6.3 8k Long-Context Accuracy Checks

We benchmark SANTA on long-context tasks borrowed from RULER [107] with a prompt length of 8192 tokens. The theoretical benefit of S^2ANTA's sparse computation is most stark in this long-context regime. With a budget of $S = 256$ in an 8192-token sequence, **S^2ANTA uses just $S/n_k = 3.125\%$ of the value-stage additions and memory accesses compared to full attention** (following Table 3.1). It achieves this while avoiding the dense value-stage weighted multiplications, leaving only the fixed normalization term counted in Table 3.1, and recovering baseline SDPA accuracy (within ±1%) across every task.

Table 3.6: Accuracy of Llama 8B on 4 long-context tasks with an 8k-token prompt. Tasks: frequent word extraction (FWE), needle-in-a-haystack (NIAH), single-hop QA (QA_1), multi-hop QA (QA_2). S: SANTA sample budget. Accuracy shows 95% bootstrap confidence intervals.

Kernel	S	FWE	NIAH	QA_1	QA_2
S^2ANTA-sys	64	96.27±0.97	94.05±1.07	71.80±3.90	67.20±4.20
S^2ANTA-sys	128	97.80±0.70	94.15±1.02	68.20±4.10	66.80±4.00
S^2ANTA-sys	**256**	**97.87±0.70**	**95.00±0.95**	**71.20±3.90**	**69.40±4.10**
S^2ANTA-strat	64	95.93±0.94	94.35±1.15	71.40±4.00	65.40±4.20
S^2ANTA-strat	128	97.73±0.80	94.30±1.10	70.80±4.20	67.40±4.20
S^2ANTA-strat	**256**	**98.33±0.60**	**94.55±1.05**	**71.20±4.00**	**67.00±4.20**
SANTA	64	92.53±1.27	93.95±1.30	64.60±4.20	63.40±4.20
SANTA	128	94.53±1.13	91.45±1.38	67.80±4.10	63.40±4.30
SANTA	256	96.20±0.93	93.15±1.18	68.00±4.10	66.40±4.10
SDPA (baseline)	**—**	**98.53±0.60**	**94.55±1.00**	**71.80±3.90**	**68.60±4.20**

For full clarity, when Llama 8B's grouped-query attention [83] with 4 grouped queries is fully accounted for, then memory access to unique rows of V can in the worst case reach 12.5%, which remains extremely sparse. Extended results with top-k comparisons are provided in Appendix Section A.2.7.

Taken together, these results establish SANTA as a controllable stochastic approximation to the value-stage attention operator: the estimator is unbiased, variance-reduced sampling recovers accuracy at useful budgets, and the theoretical benefit is largest in long-context decoding. Whereas this chapter establishes SANTA as a mathematical construct, the next chapter focuses on the systems question of how to realize the sampling and sparse accumulation efficiently on GPUs.

Chapter 4

GPU Kernels for Long-Context Stochastic Sparse Attention

Chapter 3 introduced SANTA and variance-reduced sampling rules (stratified/systematic), which specify *what* to sample to form an unbiased estimator of the value-stage aggregation AV. This chapter focuses on *how* to realize these samplers efficiently at decode time on GPUs. The emphasis is therefore no longer on defining a new estimator, but on translating the systematic S^2ANTA sampling rule into GPU kernels that parallelize sampling and sparse accumulation for long-context autoregressive decoding.

We introduce two kernel strategies, S^{2}ANTA-prop and S^{2}ANTA-flash, which differ in how they allocate samples across tiles: global proportional allocation versus speculative per-tile sampling with deferred normalization. These are GPU execution strategies for realizing the S^{2}ANTA sampling idea under tiled decode constraints rather than new attention mechanisms. S^{2}ANTA-flash uses uniform per-tile sampling with deferred tile-mass reweighting, while S^{2}ANTA-prop uses deterministic proportional integer tile budgets based on tile probability mass.

On modern GPUs, these implementation choices translate into measurable systems gains: S^2ANTA CUDA kernels achieve up to $1.5\times$ decode-step attention-kernel speedup over FlashIn-

fer and FlashDecoding at 32k-token contexts while matching baseline accuracy on long-context prompts. In batched long-context generation, the same GPU-kernel design yields up to $1.25\times$ end-to-end decode-latency speedup.

4.1 Parallelizing Stochastic Attention

Efficient implementation on GPUs presents a unique systems challenge: parallelizing the sampling control flow. Standard attention (SDPA) is amenable to "split-KV" strategies (e.g., FlashDecoding [84]) because the reduction of values is associative: partial sums can be computed locally and merged later. Sampling, however, creates a sequential dependency: determining *which* rows of V to access generally requires a global Cumulative Distribution Function (CDF). A naive implementation would require a sequential pass over the global probability distribution to draw sample indices, leaving the GPU compute units idle and preventing parallel access to the value matrix.

To make sparse value reads parallel, the kernel must know how many samples each KV tile should draw before launching the tile-local V gather. Otherwise, sampling from the global attention CDF would serialize index generation and delay all sparse V accesses until after a full global normalization/reduction. We use two strategies to break this dependency:

1. **Proportional Sample Allocation (S^2ANTA-prop):** The probability mass of each tile is pre-calculated to deterministically assign sample budgets. This requires a global barrier but ensures exact load balancing.

2. **Speculative Local Sampling (S^2ANTA-flash):** Tiles sample independently with uniform budgets, speculating that all tiles are equally important, followed by a post-hoc re-weighting. This removes the barrier but introduces sample waste.

These two strategies are developed in the following two sections and evaluated later in this chapter for decode-step kernel latency and long-context accuracy at 32k-token prompts.

4.2 Proportional Sample Allocation: S^2ANTA-Prop

S^2ANTA-prop (Figure 4.1) breaks the sequential sampling dependency by resolving the global partition function Z in a lightweight initial pass.

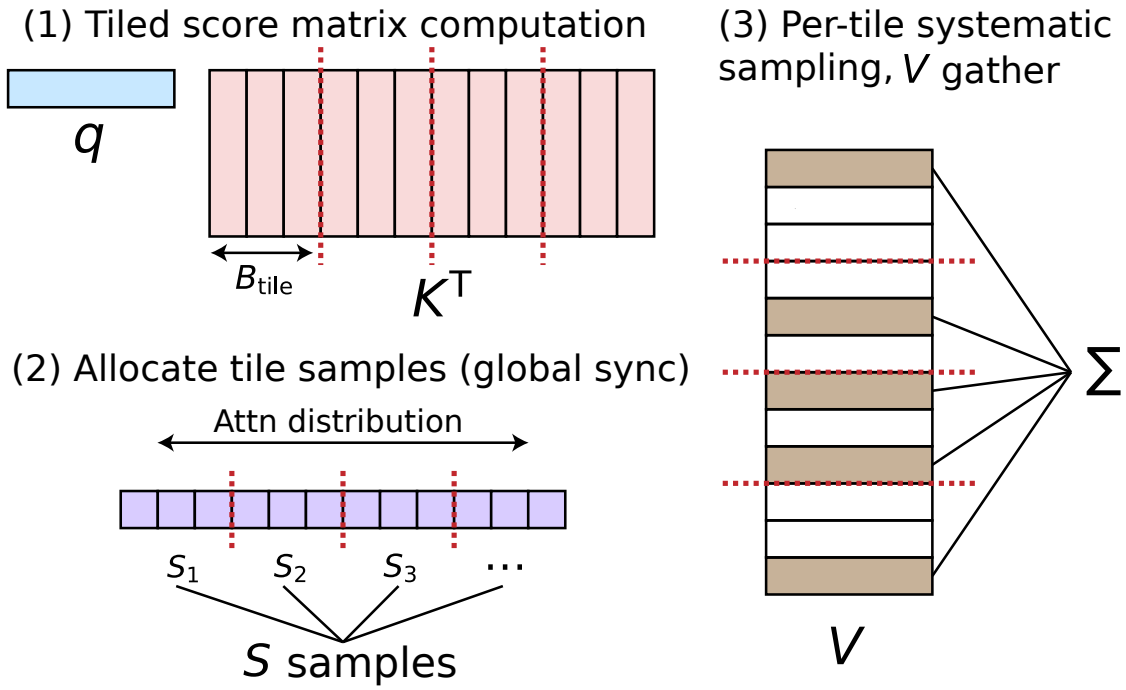


Figure 4.1: S^2ANTA-prop kernel overview.

The algorithm proceeds in three stages. The key dimension n_k is partitioned into tiles of length B_{tile}.

1. *Pass 1 (score stash & tile stats):* We compute attention scores exactly. The exponentiated scores (or a normalized equivalent) are written to global memory. Crucially, since the scores are scalars ($1 \times n_k$) while the values are vectors ($n_k \times d_k$), this stash consumes negligible bandwidth ($1/d_k$) compared to a full value fetch. We also compute and write local partition functions Z_{tile}.

2. *Global budget allocation:* A separate kernel sums the tile statistics to find the global Z. It then assigns a specific sample count to each tile: $S_{\text{tile}} \propto S \cdot (Z_{\text{tile}}/Z)$.

3. *Pass 2 (parallel gather):* Using the stashed scores and the assigned S_{tile}, each tile systematically samples indices and accumulates rows from V. Tiles with low probability mass are assigned $S_{\text{tile}} = 0$, allowing them to skip the expensive V-read entirely.

Detailed pseudocode for this implementation is provided in Appendix A.2.12.

4.3 Speculative Local Sampling: S^2ANTA-Flash

To eliminate the latency of the global synchronization barrier, S^2ANTA-flash (Figure 4.2) adopts a speculative approach structurally similar to FlashDecoding.

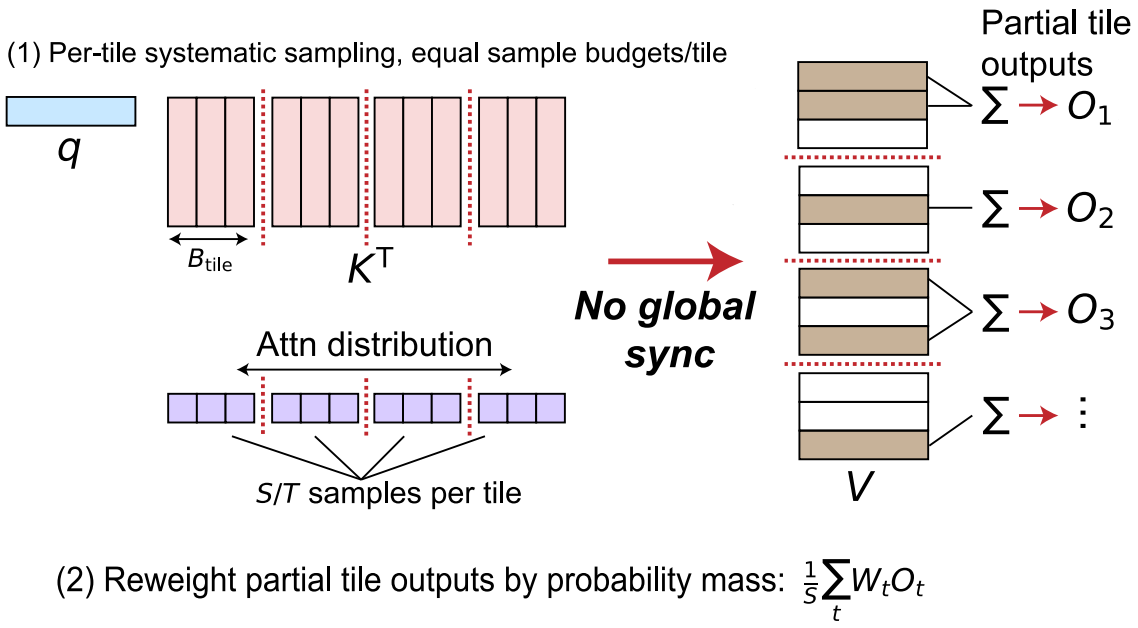


Figure 4.2: S^2ANTA-flash kernel overview.

Instead of pre-calculating the global budget, each tile is allocated a fixed, uniform budget $S_{\text{tile}} \approx S/T$, where T is the number of tiles. Each tile samples locally *as if* it contained the entire probability mass. Normalization is then deferred: a second reduction kernel computes the true global partition function Z and performs the global partition-function merge, down-weighting the partial outputs of tiles that had low probability mass.

This method maximizes parallelism by removing the global barrier but suffers from *sample waste*: tiles with low probability mass still consume compute and bandwidth to draw samples that are effectively down-weighted during the final merge. In the 32k-token evaluation discussed in the next section, matching SDPA accuracy requires substantially larger budgets for S^2ANTA-flash; the accuracy-matching operating point is $S{=}2048$ for S^2ANTA-flash versus $S{=}128$ for S^2ANTA-prop (Table 4.1). Full pseudocode for the overview, per-tile partial kernel, and merge kernel is provided in Appendix A.2.13.

4.4 Decode-Step Kernel and 32k Accuracy Benchmarks

We evaluate decode-step attention *kernel latency* together with end-task *accuracy* in the long-context decoding regime. Within this section, SDPA is used for prefill and only the *decode step* uses S^2ANTA. Kernel timings are reported for a single decoding step on an NVIDIA RTX 6000 Ada GPU (batch size 1) using bf16 Q/K/V tensors with Llama-3.1-8B-Instruct [101] tensor shapes; we compare against optimized FlashInfer and FlashDecoding backends. The measurement protocol and tensor layouts are provided in Appendix A.2.14.

To make latency comparisons meaningful, we select one operating point per kernel family as the *smallest* sample budget S that matches SDPA accuracy on 32k-token long-context tasks. This yields $S{=}128$ for S^2ANTA-prop and $S{=}2048$ for S^2ANTA-flash. These are exactly the configurations that deliver $\approx 1.5\times$ decode-step attention-kernel speedup in Fig. 4.3 while matching SDPA within confidence intervals in Table 4.1.

We next verify that these speedup points preserve long-context accuracy. We evaluate on tasks from RULER [107] using 32,768-token prompts: frequent-word extraction (FWE), needle-in-a-haystack (NIAH), single-hop QA (QA_1, derived from SQuAD [108]), and multi-hop QA (QA_2, derived from HotpotQA [109]). Results are shown in Table 4.1.

We further evaluate the same accuracy-matching operating points on LongBench v2 [110], a broader long-context reasoning benchmark. Results are shown in Table 4.2. Across three

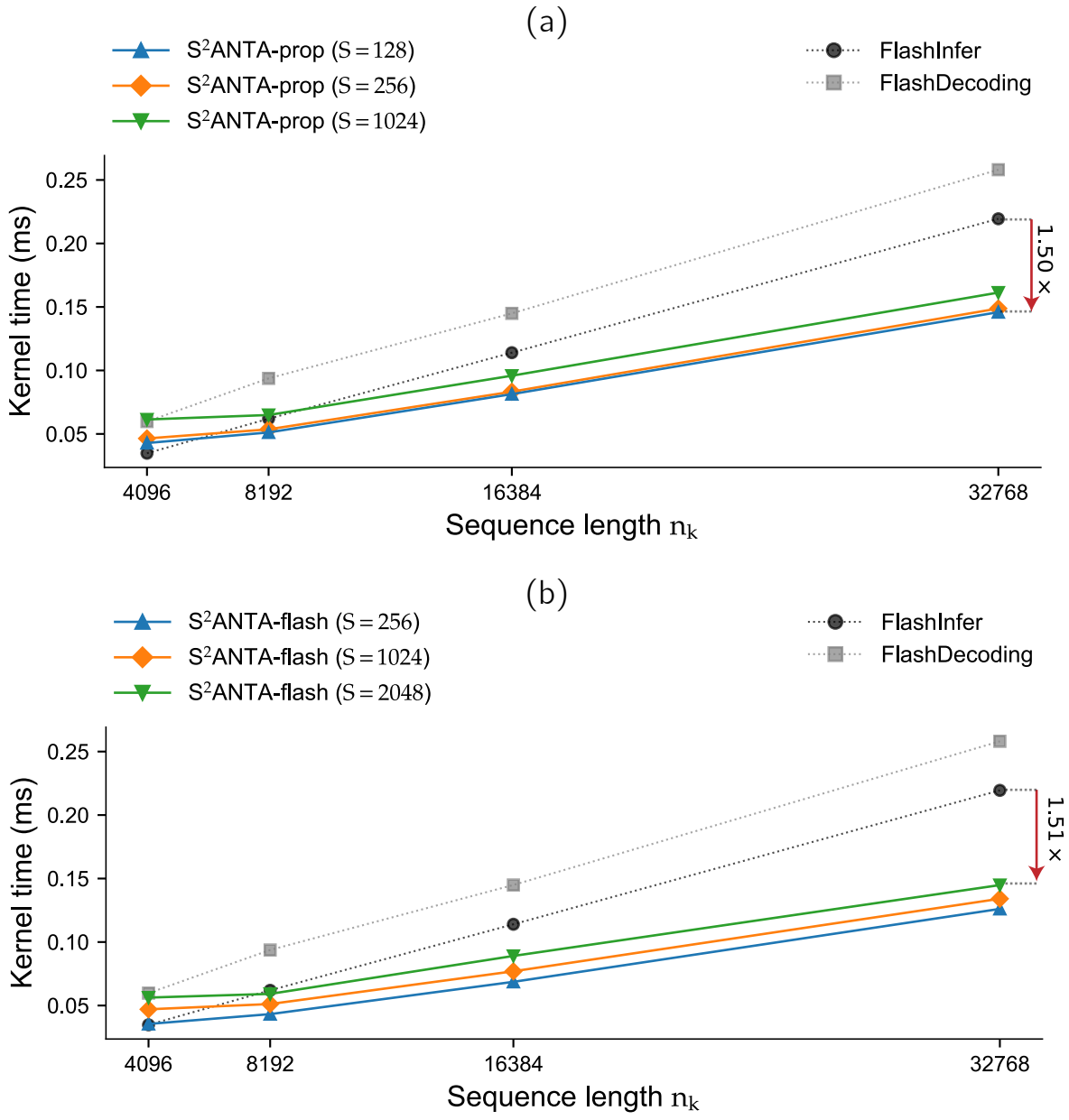


Figure 4.3: S^2ANTA kernel latency on Llama 8B tensor shapes for 1 decoding step. (a) S^2ANTA-prop demonstrates a 1.50× speedup relative to FlashInfer at a 32k-token context. (b) S^2ANTA-flash similarly exhibits a 1.51× speedup. Both operating points ($S = 128$ for prop, $S = 2048$ for flash) correspond to configurations that recover baseline accuracy (Table 4.1).

Table 4.1: Accuracy of Llama 8B on 4 long-context tasks with a 32,768-token prompt. SDPA is used in prefill. Tasks: frequent word extraction (FWE), needle-in-a-haystack (NIAH), single-hop QA (QA_1), multi-hop QA (QA_2). S: sample budget. Accuracy shows 95% bootstrap confidence intervals.

Kernel	S	FWE	NIAH	QA_1	QA_2
S²ANTA-prop	**128**	**95.40±1.10**	**98.25±0.62**	**64.40±4.50**	**60.20±4.20**
S^2ANTA-prop	256	95.47±1.10	98.50±0.57	63.40±4.10	60.60±4.00
S^2ANTA-prop	1024	95.40±1.07	98.40±0.60	64.20±4.00	59.60±4.40
S^2ANTA-flash	256	66.20±2.40	88.95±1.43	63.00±4.20	57.20±4.30
S^2ANTA-flash	1024	91.60±1.43	98.10±0.62	62.20±4.10	61.00±4.30
S²ANTA-flash	**2048**	**94.13±1.13**	**98.25±0.62**	**64.60±4.20**	**60.00±4.40**
SDPA (baseline)	**—**	**95.60±1.00**	**98.35±0.62**	**64.00±4.10**	**58.80±4.10**

runs, prompts longer than 32k tokens are truncated to 32k following the repository protocol. In this 30k-token regime, S^2ANTA-prop with $S{=}128$ and S^2ANTA-flash with $S{=}2048$ match SDPA within bootstrap confidence intervals, using the same operating points where Fig. 4.3 reports $\approx 1.5\times$ decode-step attention-kernel speedup.

At these operating points, the speedups are driven primarily by reduced memory traffic from sparsifying reads of the value matrix V. For S^2ANTA-prop, $S{=}128$ corresponds to extremely sparse value access at 32k contexts; even accounting for the worst-case union across Llama 8B's four-query GQA group, this is $\lesssim 1.56\%$ of value rows.

S^2ANTA-flash attains comparable speedup, but requires a larger total sample budget because uniform per-tile sampling expends samples on low-mass tiles before the deferred renormalization step ("sample waste").

The full microbenchmark protocol, additional 8k-token kernel-accuracy results, full LongBench v2 budget sweep, and same-state fidelity/tile-sensitivity analyses are available in Appendices A.2.14, A.2.15, A.2.8, and A.2.17, respectively.

Table 4.2: LongBench v2 accuracy for Llama 8B at the main accuracy-matching operating points. S: sample budget. Accuracy shows 95% bootstrap confidence intervals.

Kernel	S	Acc. (%)	Avg. scored tok.
S²ANTA-prop	**128**	**28.098±1.028**	**29998.4**
S²ANTA-flash	**2048**	**28.827±3.422**	**29996.4**
SDPA (baseline)	**—**	**28.363±1.509**	**29996.0**

These timings isolate the decode-step attention kernel. The next section examines whether the kernel-level gain survives integration into a full generation loop, where prefill, MLPs, normalization, projections, and framework overheads remain unchanged.

4.5 End-to-End Batched Decoding and Energy Considerations

We next measure end-to-end decode latency by integrating our kernels into a batched generation loop. The experiment uses Llama-3.1-8B-Instruct on an NVIDIA RTX 6000 Ada GPU with bf16 model weights and KV cache. We use representative QA prompts truncated to 30k tokens, batch size 6, and decode 2048 new tokens per prompt. Prefill is kept dense for all methods; S^2ANTA is applied only during autoregressive decoding. We use the same accuracy-matching operating points as in Section 4.4: $S = 2048$ for S^2ANTA-flash and $S = 128$ for S^2ANTA-prop.

Table 4.3: End-to-end batched generation latency for Llama-3.1-8B-Instruct on RTX 6000 Ada. Prompts are truncated to 30k tokens, batch size is 6, and each prompt decodes 2048 new tokens. Prefill remains dense; S^2ANTA is applied only during decode. Results are averaged over 9 batches.

Method	S	Prefill (s/batch)	Decode (s/batch)	ms / output token	Decode speedup
FlashAttn-2	–	56.69	101.22	8.237	1.000×
S^2ANTA-flash	**2048**	**56.61**	**80.97**	**6.589**	**1.250×**
S^2ANTA-prop	**128**	**57.03**	**82.36**	**6.702**	**1.229×**

Table 4.3 shows that the decode-step kernel gains translate to end-to-end generation: S^2ANTA-flash achieves a $1.25\times$ decode-latency speedup and S^2ANTA-prop achieves a $1.229\times$ speedup relative to FlashAttention-2 [111]. The prefill times remain essentially unchanged, as expected, since these measurements use dense prefill and replace only the decode-time softmax–V aggregation. Thus, SANTA is best viewed as a specialized backend for long-context, decode-heavy workloads rather than a prefill accelerator.

The observed end-to-end speedup is consistent with a simple Amdahl-style bandwidth

model. SANTA modifies only the decode-time softmax–V aggregation, so its realized benefit is bounded by the share of decode bandwidth spent on the KV pathway. In bf16, for one batched decode step, the Llama 8B weights contribute roughly 15 GB of traffic, while the KV cache for a batch of six 30k-token prompts contributes roughly 24 GB. Combining this bandwidth share with the measured long-context attention-kernel speedup $s_{\mathrm{KV}} \approx 1.5$ predicts $1/((15/39)+(24/39)/1.5) \approx 1.26\times$, close to the observed $1.25\times$ decode speedup in Table 4.3. This also explains why longer contexts and larger batches should increase SANTA's marginal benefit, while shorter contexts, prefill-heavy workloads, or aggressive KV-cache quantization may reduce it; model-weight quantization may have the opposite effect by increasing the relative share of KV traffic.

Our measured kernel and end-to-end speedups are on RTX 6000 Ada GPUs. Because SANTA reduces KV-cache traffic, we expect the qualitative benefit to remain relevant in bandwidth-bound decode on datacenter GPUs such as A100/H100, but exact speedups will depend on memory bandwidth, cache hierarchy, batching, and kernel integration.

Beyond latency, S^2ANTA's design offers two complementary paths to energy reduction. First, sparse memory access reduces value-cache traffic from n_k to S rows per query; at accuracy-matching operating points, this corresponds to >90% reductions in value-stage memory access at 32k contexts. Second, replacing the dense value-stage weighted multiply-accumulate with sampled gather-and-add removes the multiplications that scale with n_k, leaving only the fixed d_k-dimensional normalization counted in Table 3.1. While our current GPU kernels primarily exploit the first benefit, hardware architectures optimized for sparse, adder-centric computation could realize both. These characteristics make S^2ANTA a natural fit for emerging AI accelerators where memory bandwidth and multiplier energy are primary constraints.

A more detailed comparison with top-k attention, including the distinction between matched-budget comparisons and iso-compute comparisons, is deferred to Appendix A.2.7.

4.6 Bernoulli $qK^\top$ Sampling for Sparse Key Access

While the preceding SANTA kernels target the post-softmax value aggregation stage, Bernoulli $qK^\top$ sampling is a complementary inference-time technique for the score stage: it represents query elements as Bernoulli variables to form sparse ternary queries in $\{-1, 0, +1\}$, yielding an unbiased estimator of $qK^\top$ and inducing feature-sparse key access during decoding. The implemented kernels and measured speedups in the preceding sections are driven primarily by value-stage sparsification; Bernoulli $qK^\top$ is presented here as a complementary mechanism that can be combined with SANTA to sparsify both branches.

SANTA reduces memory access for the value matrix by sampling from the softmax distribution; however, it leaves the score computation ($qK^\top$) intact. As illustrated in Fig. 4.4, Bernoulli sampling of the query naturally zeroes out query-irrelevant features, thereby effectively reducing memory access for the key matrix.

4.6.1 Bernoulli $qK^\top$ Estimator and BitNet Results

We propose representing each query magnitude as an independent Bernoulli probability, allowing the corresponding key feature to be pruned when the sampling probability is low. While this method is applicable to the prefill phase, it induces sparse key access specifically during

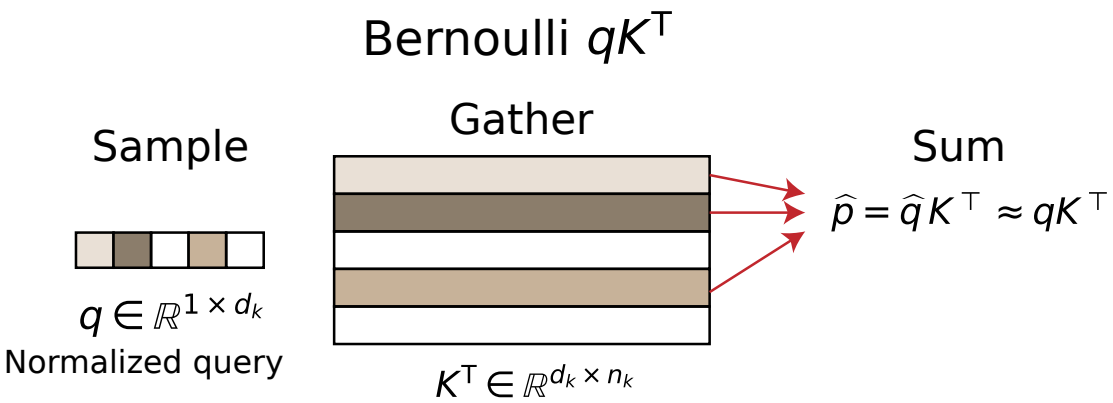


Figure 4.4: Bernoulli $qK^\top$ represents query magnitudes as Bernoulli probabilities and carries the sign in a sparse ternary query vector, sparsifying along the feature dimension d_k with multiplier-free arithmetic.

decoding; therefore, we focus on a single query q. First, the query vector is normalized as $q \leftarrow q/\text{norm}$, where $\text{norm} \geq \max_i |q_i|$. This ensures that $|q_i| \in [0, 1]$ can be interpreted as a Bernoulli probability, with $\text{sgn}(q_i)$ carried separately in the ternary sampled query.

Next, we estimate the product $p = qK^\top$ as:

$$\widehat{p} = \frac{\text{norm}}{B} \sum_{n=1}^{B} \hat{q}^{(n)} K^\top, \tag{4.1}$$

where B is the sample count and $\hat{q}^{(n)}$ is a sparse ternary vector constructed via Bernoulli sampling. A more detailed construction of $\hat{q}^{(n)}$, together with the Bernoulli $qK^\top$ derivation and error analysis, is given in Appendix A.2.5. The linearity of expectation ensures this estimator is unbiased. Because $\hat{q}^{(n)}$ contains only values in $\{-1, 0, +1\}$, $\widehat{p}$ can be computed without multiplications by gathering and accumulating selected rows of $K^\top$, followed by normalizing bit shifts for fixed-point representations.

Table 4.4 shows GSM8K results for Bernoulli $qK^\top$ applied to BitNet 2B [112] for both prefill and decoding. Using $B = 4$ stratified samples nearly recovers the accuracy of standard SDPA for the GSM8K benchmark, although the approximated scores typically have a 30% error, as discussed in Appendix A.2.5. This suggests that the $qK^\top$ product within the highly quantized BitNet 2B architecture can tolerate numerical errors while maintaining strong overall performance.

Crucially, Bernoulli $qK^\top$ enables sparse access to cached keys, a benefit that is uniquely effective during decoding since the prefill phase requires nearly all features. This is shown in Table 4.4: with $B = 4$, the model needs only 67.5% of key features per head, compared to $\sim 100\%$ for prefill, while maintaining accuracy within 2% of the SDPA baseline.

4.6.2 Mean Group Query for GQA

While Bernoulli $qK^\top$ effectively prunes irrelevant features for individual heads, Grouped Query Attention (GQA) [83] complicates this by sharing keys and values across multiple heads, such

Table 4.4: Performance of Bernoulli $qK^\top$ (stratified) on GSM8K (BitNet 2B) with the test set evaluated three times, and confidence intervals obtained by bootstrapping.

Method	**B**	**Acc. (%)** (95% CI)	**Avg. Tok.**	$K^\top$ **Access (%)** Head	Group
SDPA	–	65.7 ± 1.5	306.1	100	100
Bernoulli $qK^\top$	1	1.9 ± 0.43	1478.6	25.6	65.8
	2	48.7 ± 1.4	345.9	46.3	88.0
	4	64.5 ± 1.5	309.4	**67.5**	**97.9**
	8	65.1 ± 1.5	316.9	82.6	99.8
	16	65.8 ± 1.5	317.7	91.1	100
Mean Group Query Bernoulli $qK^\top$	2	48.6 ± 1.5	318.6	57.9	57.9
	4	63.8 ± 1.5	311.4	**84.7**	**84.7**
	8	65.6 ± 1.6	311.4	97.1	97.1
	16	66.2 ± 1.5	310.2	99.7	99.7

as groups of 4 in BitNet. Consequently, the set of accessed features becomes the union of those required by all queries in the group, significantly increasing memory traffic. Table 4.4 illustrates this effect: with $B = 4$ samples, $K^\top$ access jumps from 67.5% per head to 97.9% for the group. To preserve sparsity in this scenario, we propose using a representative query m for the entire group, defined as the mean of the queries' absolute values. We then construct the estimator $\widehat{m}$ for the mean query m via Monte Carlo sampling, detailed in Appendix A.2.5, which stochastically prunes the low-amplitude mean features.

With $B = 4$ samples, $\widehat{m}$ contains approximately 15% of zero elements, which allows fetching only 85% of key features for the group, as shown in Table 4.4. The product $p = qK^\top$ for each query is finally estimated as:

$$\widehat{p} = \left(\widehat{m} \odot \frac{q}{m}\right) K^\top, \tag{4.2}$$

where $\odot$ denotes element-wise multiplication. In this scenario, calculating $\widehat{p}$ requires normalization by m to compensate for the bias introduced by the mean group query. As shown in Table 4.4, the resulting estimator achieves similar accuracy to the single-query configuration while saving 15% of key memory accesses. We employ the mean group query estimator in

Eq. 4.2 in all the following experiments.

4.6.3 Long-Context Results for Bernoulli $qK^\top$

We evaluate Bernoulli $qK^\top$ on long-context tasks using Llama 8B, applying the estimator strictly during decoding since prefill requires full feature access. As detailed in Table 4.5, using $B = 8$ samples reduces key feature access to 72.8% for Llama 8B, with minimal accuracy loss, less than 2% on NIAH, QA_1, and QA_2. Furthermore, this experiment highlights that key memory access patterns vary significantly across models. In contrast to Llama 8B, BitNet 2B requires accessing 97.1% of features for the same budget, $B = 8$, as shown in Table 4.4, a difference we attribute to distinct query distributions.

4.6.4 Combining Score-Stage and Value-Stage Sampling

Combining SANTA with Bernoulli $qK^\top$ enables sparse key–value accesses during decoding while replacing the majority of multiplications with additions. Table 4.6 shows the accuracy for the GSM8K benchmark using BitNet 2B and $B = 4$ Bernoulli samples for the $qK^\top$ computation, as the number of SANTA samples S is varied for the value stage. Using $B = 4$ for queries and $S = 64$ for attention weights almost recovers the SDPA baseline with less than 2% accuracy drop. Critically, this combined approach reduces key access by 15.3% and value access by 89.2%.

Table 4.5: Accuracy of Llama 8B on 4 long-context tasks with a 4,096-token prompt. Tasks: frequent word extraction (FWE), needle-in-a-haystack (NIAH), single-hop QA (QA_1), multi-hop QA (QA_2). B: Bernoulli $qK^\top$ (mean group query) sample budget. Accuracy shows 95% bootstrap confidence intervals.

Kernel ($qK^\top$)	B	$K^\top$ Access (%)	FWE	NIAH	QA_1	QA_2
Bernoulli	4	47.3	70.4±2.0	87.2±1.9	72.0±3.8	66.2±4.1
Bernoulli	**8**	**72.8**	89.4±1.4	**93.8±1.4**	**73.6±3.9**	**70.2±3.9**
Bernoulli	16	92.1	92.7±1.2	94.5±1.2	74.8±3.5	71.6±4.2
SDPA	–	–	**94.0±1.1**	**95.0±1.2**	**73.8±3.8**	**70.8±4.0**

Table 4.6: Performance of Bernoulli $qK^\top$ (mean group query) + SANTA on GSM8K (BitNet 2B) with the test set evaluated three times, and confidence intervals computed by bootstrapping.

Config	**Acc. (%)** (95% CI)	**Avg. Tok.**	**(K,V) Access (%)** Group
SDPA	65.7 ± 1.5	306.1	(100, 100)
$B = 4,\ S = 8$	55.0 ± 1.6	319.2	(84.7, 2.4)
$B = 4,\ S = 16$	60.7 ± 1.5	313.3	(84.7, 3.7)
$B = 4,\ S = 32$	62.3 ± 1.5	308.4	(84.7, 6.2)
B = 4, S = 64	63.8 ± 1.5	304.6	**(84.7, 10.8)**
$B = 4,\ S = 128$	64.1 ± 1.5	303.2	(84.7, 14.3)

4.7 Discussion, Limitations, and Hardware Outlook

Chapter 3 positioned SANTA relative to exact attention kernels, KV-cache compression and selection, score-stage approximations, and deterministic truncation. The systems result in this chapter is narrower and more concrete: after dense prefill and exact score computation, S^2ANTA kernels reduce decode-time value-cache traffic by sampling rows of V and aggregating them with gather-and-add. This is the source of the measured kernel and end-to-end speedups. The proportional and flash variants differ in how they make this sampling GPU-parallel, but both instantiate the same basic value-stage SANTA estimator developed in Chapter 3.

The main limitation is that SANTA is specialized for long-context, decode-heavy workloads. It is not a prefill accelerator, and it does not claim to dominate deterministic truncation or selection in every regime. Its realized benefit is bounded by the share of decode time and decode bandwidth spent on the KV/value pathway. Shorter contexts, prefill-heavy workloads, aggressive KV-cache quantization, or architectures that already reduce KV bandwidth may reduce its marginal benefit; longer contexts and larger batches should increase it. The measured speedups in this chapter were obtained on RTX 6000 Ada GPUs, so exact performance on other accelerators will depend on memory bandwidth, cache hierarchy, batching, and kernel integration.

Bernoulli $qK^\top$ sampling should be read as a secondary score-stage extension rather than the main result of the chapter. The Bernoulli experiments suggest that stochastic ternary queries can

expose sparse key-feature access, and Table 4.6 shows that this direction can be combined with value-stage SANTA. However, the headline SANTA speedups reported earlier in the chapter are driven primarily by value-stage sparsification. A complete sparse-key system would require additional model- and hardware-dependent work, especially under grouped-query attention and model-specific query distributions.

Overall, the contribution of Chapters 3–4 is that stochastic sampling can be turned into a practical long-context attention backend: the estimator is unbiased, variance-reduced sampling recovers accuracy at useful budgets, and GPU kernels convert sparse value-row access into measured decode acceleration. From a hardware perspective, this points toward sparse, stochastic, and adder-centric inference systems in which not every value-cache row needs to be read during the value aggregation at every decode step. The next chapter asks an exploratory question on the feed-forward network (FFN) side: whether stochastic neuron gates can expose skippable down-projection weight reads in transformer feedforward networks.

Chapter 5

Stochastic Neurons for Sparse Transformer Feedforward Networks

5.1 Motivation and Overview

The preceding chapters applied stochastic sampling to attention by sampling value-cache rows during long-context decoding. This chapter turns to the other large sublayer in a transformer block: the feed-forward network. Transformer architectures alternate attention sublayers with position-wise feed-forward sublayers [41]. In each FFN block, an input vector is expanded into a much larger intermediate representation and then projected back down to the model width. During autoregressive inference, especially at small batch size, these projections repeatedly stream large weight matrices through the GPU memory hierarchy. The question here is whether the sampling object can be changed from attention rows to FFN channels: instead of reading all down-projection weights for every token, can stochastic gates expose a smaller active set of channels whose weights must be read?

Activation sparsity is one natural way to expose this kind of memory saving. If an intermediate FFN channel is exactly zero, then that channel contributes nothing to the following down projection. In principle, the corresponding down-projection weight vector does not

need to be read for that token. This is straightforward for activations such as ReLU, which produce exact zeros deterministically and were studied as sparse hidden representations in deep rectifier networks [113]. Recent LLM work has also revisited activation sparsity and contextual sparsity as routes to efficient inference [114, 115]. However, smooth dense alternatives to ReLU, including SiLU/Swish and SwiGLU [116, 117, 118], do not naturally produce exact zeros. These activations can be attractive for model quality, but they remove the deterministic sparsity exposed by ReLU.

This chapter studies a simple stochastic replacement for SiLU-style gates. For a scalar preactivation x, deterministic SiLU computes $x\sigma(x)$, where σ is the logistic sigmoid [116, 117]. The stochastic counterpart samples a Bernoulli gate,

$$B \sim \text{Bernoulli}(\sigma(x)), \qquad \tilde{a}(x) = xB. \tag{5.1}$$

This activation is exactly zero whenever $B = 0$, but it preserves deterministic SiLU in pointwise expectation:

$$\mathbb{E}[\tilde{a}(x)] = x\mathbb{E}[B] = x\sigma(x) = \text{SiLU}(x). \tag{5.2}$$

Thus, the stochastic gate creates exact activation zeros without changing the local expected value of the SiLU neuron.

The goal of this chapter is not to claim that this replacement already improves full end-to-end language-model inference. Rather, the goal is to test whether stochastic SiLU/SwiGLU is a plausible interface between model adaptation and sparse FFN kernels. The empirical question has three parts. First, can a pretrained language model be adapted to tolerate stochastic gates without full pretraining? Second, do the resulting zeros create a meaningful memory-read opportunity in the FFN down projection? Third, can a sparse kernel exploit the zeros in an isolated FFN setting?

The main deployment-motivated experiment retrofits Qwen2.5-0.5B from the Qwen model family [119, 120] with stochastic SwiGLU gates using LoRA [121] on GSM8K [102]. The

base model is frozen, every transformer MLP gate activation is replaced by the stochastic counterpart, and only low-rank adapters are trained. This experiment is the most practical setting in the chapter because adapting an existing pretrained model is much cheaper than training a new language model from scratch. The best setting, rank-256 LoRA applied to both FFN and attention projections, recovers base-level GSM8K generated-answer exact match under stochastic inference while maintaining approximately 59% exact zeros.

A complementary experiment trains a 707M-parameter GPT-style model [122] from scratch on FineWeb-Edu [123] with stochastic SiLU and a straight-through estimator. This is not presented as the intended deployment path. Instead, it answers a basic stability question: whether the stochastic activation itself can support language-model training at moderate scale, or whether the LoRA experiment is merely compensating for an unusable activation. The stochastic model trains stably for 2B tokens and produces high sparsity, although it converges to slightly worse validation and test loss than the deterministic SiLU baseline, representing a sparsity vs. performance tradeoff.

Finally, the chapter evaluates whether the exact zeros can be converted into a real FFN-level speedup. A standalone sparse FFN microbenchmark using real trained checkpoint weights shows a 1.309× speedup over the corresponding dense stochastic FFN implementation. This result is intentionally scoped: it shows that the sparse down-projection operation can benefit from the induced zeros, but the current implementation does not yet demonstrate full-model end-to-end inference speedup.

5.2 Stochastic SiLU and SwiGLU Gates

Figure 5.1 summarizes the mechanism.

The stochastic activation in Eq. 5.1 is applied elementwise. For a vector of preactivations

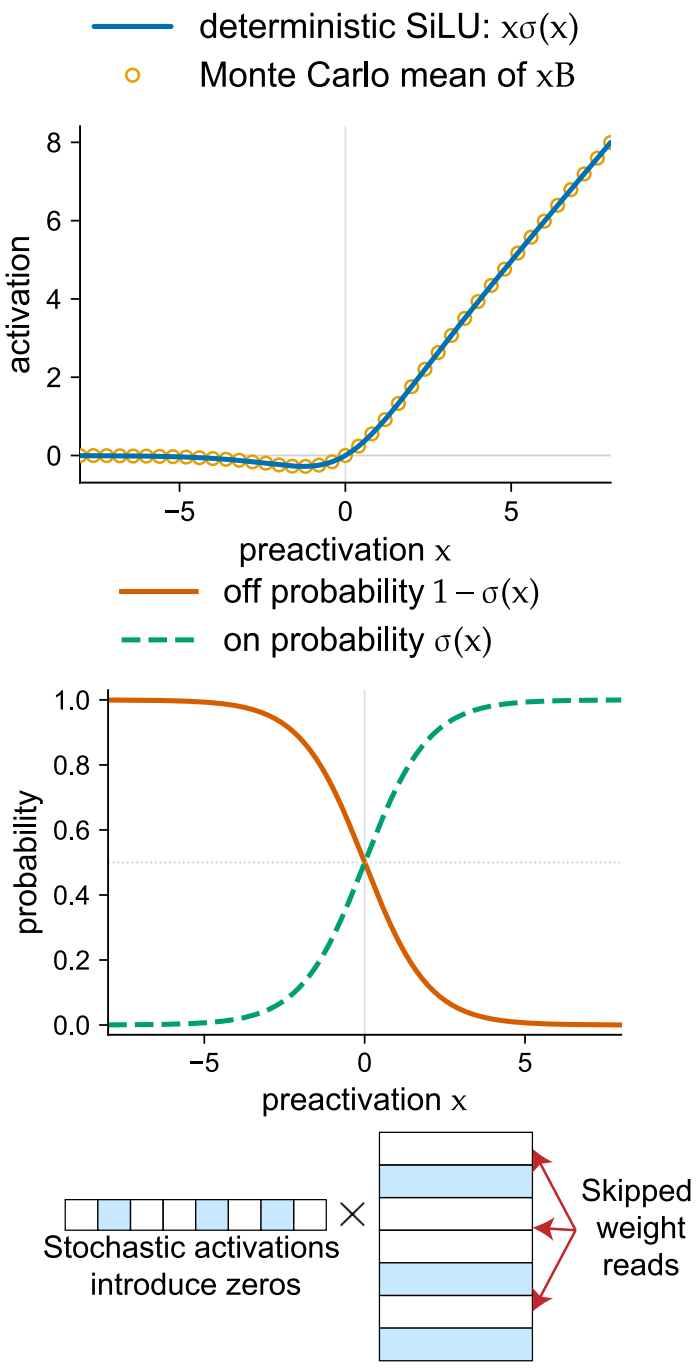


Figure 5.1: **Expectation-matched stochastic SiLU creates exact zeros.** The stochastic activation $\tilde{a} = xB$, with $B \sim \text{Bernoulli}(\sigma(x))$, matches deterministic SiLU $x\sigma(x)$ in pointwise expectation while producing exact zeros with probability $1 - \sigma(x)$. In a transformer FFN, zero intermediate channels expose down-projection weight reads that can in principle be skipped.

x, each coordinate samples an independent Bernoulli gate with probability $\sigma(x_i)$ and outputs

$$\tilde{\mathbf{a}}(\mathbf{x})_i = x_i B_i, \qquad B_i \sim \text{Bernoulli}(\sigma(x_i)). \tag{5.3}$$

The local expectation match holds coordinatewise:

$$\mathbb{E}[\tilde{\mathbf{a}}(\mathbf{x})_i] = x_i \sigma(x_i). \tag{5.4}$$

This should be interpreted as a local activation-level property, not as a claim that the entire stochastic network is an unbiased estimator of the deterministic network. Once stochastic activations are composed across many nonlinear transformer layers, the full model output distribution is not guaranteed to match the deterministic model. The purpose of the expectation match is to make the local replacement as close as possible to the original SiLU gate while still creating exact zeros.

For training and adaptation, we use a straight-through estimator, following the standard approach of using a sampled or hard forward value with a surrogate backward gradient for stochastic neurons [124]. The forward pass uses the sampled activation xB, while the backward pass follows the deterministic SiLU surrogate. In stop-gradient notation, the training-time activation can be written as

$$\tilde{a}_{\text{STE}}(x) = \text{SiLU}(x) + \text{sg}(xB - \text{SiLU}(x)), \tag{5.5}$$

where $\text{sg}(\cdot)$ denotes stop-gradient. Therefore, the value used by the forward computation contains exact zeros, while gradients flow as if the activation were ordinary SiLU.

This replacement also applies naturally to SwiGLU FFNs [118]. A SwiGLU-style feed-forward block can be written as

$$\text{FFN}(\mathbf{h}) = W_d \left[\text{SiLU}(W_g \mathbf{h}) \odot (W_u \mathbf{h})\right], \tag{5.6}$$

where W_g is the gate projection, W_u is the up projection, W_d is the down projection, and $\odot$ denotes elementwise multiplication. In the stochastic version, the SiLU gate branch is replaced by

$$\widetilde{\mathrm{SiLU}}(W_g\mathbf{h}) = (W_g\mathbf{h}) \odot \mathbf{B}, \qquad B_j \sim \mathrm{Bernoulli}\left(\sigma((W_g\mathbf{h})_j)\right). \tag{5.7}$$

The stochastic intermediate channel is then

$$\tilde{\mathbf{z}} = \left[(W_g\mathbf{h}) \odot \mathbf{B}\right] \odot (W_u\mathbf{h}). \tag{5.8}$$

Whenever $B_j = 0$, the j-th intermediate channel is exactly zero. Since the final projection is linear, that channel contributes nothing to $W_d\tilde{\mathbf{z}}$. The systems opportunity is therefore concentrated in the down projection: if the implementation knows which intermediate channels are zero, it can skip the corresponding down-projection weight reads.

Throughout this chapter, we use a deliberately conservative memory model. The projections required to produce the stochastic mask remain dense. In a SwiGLU FFN, the gate and up projections are still evaluated normally. In a two-matrix SiLU FFN, the first projection is still evaluated normally. Only the final FFN projection is treated as skippable. This makes the memory estimates conservative, but it also avoids claiming savings from operations that the current experiments do not skip.

The stochastic gate is also related to prior work on stochastic neurons and sampling-based neural-network computation. Earlier work on stochastic neurons and straight-through estimators studied how to train networks with sampled binary units [124], and recent work has revisited sampling inside feed-forward neural networks as a way to trade stochastic computation against model behavior [125]. The construction in this chapter differs in its target and use: the sampled binary gate multiplies a real-valued preactivation inside a transformer FFN, is chosen to match SiLU in pointwise expectation, and is used to expose exact zeros for sparse down projections.

5.3 Retrofitting a Pretrained Language Model with Stochastic Gates

The central experiment in this chapter asks whether an existing pretrained language model can be adapted to stochastic FFN gates without full retraining. This is the most realistic deployment setting for the method. Training a new language model from scratch is expensive, while parameter-efficient adaptation can be used to test whether the pretrained model has enough flexibility to tolerate a new stochastic activation rule.

Qwen2.5-0.5B is used as the pretrained base model [119, 120] and GSM8K is used as the downstream reasoning task [102]. The deterministic baseline is the unmodified base model with ordinary deterministic SwiGLU activations and no LoRA adapter. For the stochastic settings, the original model weights are frozen, every transformer MLP gate activation is replaced by the stochastic SwiGLU gate in Eq. 5.7, and LoRA adapters are trained [121]. The stochastic forward pass is used during adaptation and evaluation, while gradients through the stochastic gate are supplied by the SiLU straight-through surrogate in Eq. 5.5.

The LoRA sweep varies two factors. The first is adapter rank, with $r \in \{64, 128, 256\}$. The second is adapter placement. In the FFN-only setting, adapters are applied to the gate, up, and down projections of each MLP. In the FFN+attention setting, adapters are also applied to the query, key, value, and output projections of attention. This sweep tests whether adapting only the FFN is sufficient, or whether the attention pathway also needs to adjust to the changed FFN statistics.

The training targets are generated solution traces from the deterministic base model. For each GSM8K training problem, the deterministic teacher samples multiple candidate solutions; candidates whose extracted numeric answer matches the gold answer are retained, and the shortest correct traces are selected as supervised targets. The stochastic student is then trained by next-token cross entropy on these selected teacher traces.

Evaluation separates task success from distributional diagnostics. The primary task metric

is generated-answer exact match on the 1,319-example GSM8K test split. Token-level output sampling is disabled during evaluation, so variation across repeated stochastic runs comes from the activation masks, not from sampling the next-token distribution. Exact match is reported as mean ± standard deviation over 10 stochastic activation-mask repeats. Teacher-forced cross entropy and perplexity are also reported on the official GSM8K gold solutions. These CE/PPL values are useful diagnostics, but they are not the training objective and should not be read as the primary task result. Zero fractions are measured during the same fixed gold-solution teacher-forced pass. Appendices A.3.1 and A.3.2 give qualitative examples and the Qwen protocol.

Table 5.1: **Qwen2.5-0.5B GSM8K LoRA retrofit under stochastic SwiGLU gates.** Generated-answer exact match is the primary task metric on the 1,319-example GSM8K test split and is reported as mean ± standard deviation over 10 stochastic activation-mask repeats. CE/PPL and zero activations are auxiliary teacher-forced diagnostics on the official gold solutions.

Model setting	Adapted modules	Rank	Trainable adapter params	Generated-answer EM (%)
Deterministic base, no LoRA	**none**	**–**	**–**	**13.12 ± 0.00**
Stochastic gates + LoRA	FFN projections	64	26.54M	4.31 ± 0.38
Stochastic gates + LoRA	FFN projections	128	53.08M	6.20 ± 0.40
Stochastic gates + LoRA	FFN projections	256	106.17M	9.19 ± 0.72
Stochastic gates + LoRA	FFN + attn projections	64	35.19M	7.97 ± 0.71
Stochastic gates + LoRA	FFN + attn projections	128	70.39M	10.81 ± 0.71
Stochastic gates + LoRA	**FFN + attn projections**	**256**	**140.77M**	**13.44 ± 0.52**

Model setting	Adapted modules	Rank	Gold-solution CE	Gold-solution PPL	Zero activations (%)
Deterministic base, no LoRA	**none**	**–**	**0.804**	**2.234**	**–**
Stochastic gates + LoRA	FFN projections	64	1.430	4.178	61.24%
Stochastic gates + LoRA	FFN projections	128	1.360	3.895	60.80%
Stochastic gates + LoRA	FFN projections	256	1.345	3.839	59.65%
Stochastic gates + LoRA	FFN + attn projections	64	1.333	3.792	60.40%
Stochastic gates + LoRA	FFN + attn projections	128	1.329	3.778	59.57%
Stochastic gates + LoRA	**FFN + attn projections**	**256**	**1.501**	**4.485**	**58.69%**

Table 5.1 and Figure 5.2 show a clear dependence on adapter capacity and adapter scope. FFN-only LoRA improves as rank increases, but even rank 256 remains below the deterministic base on generated-answer exact match. Adding attention adapters substantially improves recovery at every rank. The strongest setting, rank-256 FFN+attention LoRA, reaches $13.44 \pm 0.52\%$ GSM8K exact match, compared with 13.12% for the deterministic base. This is the central positive result of the chapter: a pretrained language model can be adapted to run with stochastic SwiGLU gates while recovering base-level generated-answer accuracy on this task.

The result should also be interpreted carefully. The best recovery uses a relatively large adapter, with 140.77M trainable LoRA parameters. Moreover, teacher-forced CE and PPL remain worse than the deterministic base even when exact match recovers. The rank-256 FFN+attention setting therefore demonstrates task-level recovery under stochastic inference, not full distributional equivalence to the deterministic model. This distinction is important: stochastic activations may preserve enough behavior for a downstream task after adaptation while still changing the model's token-level probabilities.

The stochastic sparsity is substantial. In the best Qwen setting, the overall zero-activation rate during the fixed gold-solution pass is 58.69%. Figure 5.3 shows that this sparsity is present throughout the transformer stack, with layerwise zero rates ranging from 41.37% to 69.19%. This is the sparsity level used for the Qwen memory-read estimate in Table 5.3.

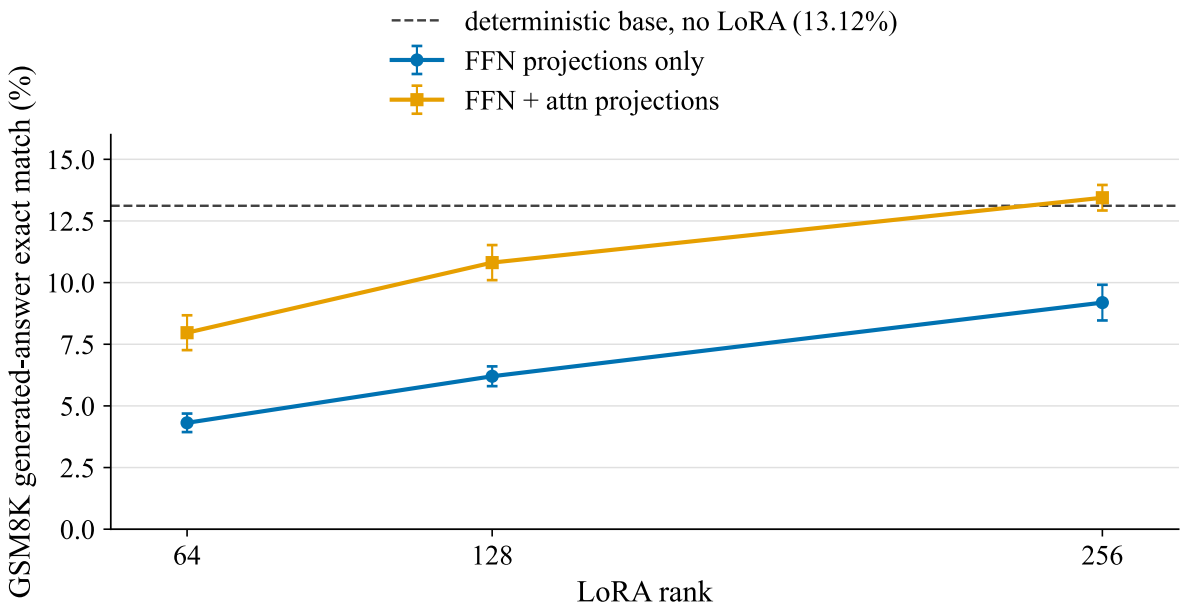


Figure 5.2: **Adapter capacity recovers generated-answer accuracy under stochastic gates.** With stochastic SwiGLU gates enabled, FFN-only LoRA improves with rank but remains below the deterministic no-LoRA base. Adapting both FFN and attention projections improves recovery, with rank-256 FFN+attention LoRA reaching 13.44 ± 0.52% GSM8K generated-answer exact match compared with 13.12% for the deterministic base. Error bars show standard deviation over 10 stochastic activation-mask repeats.

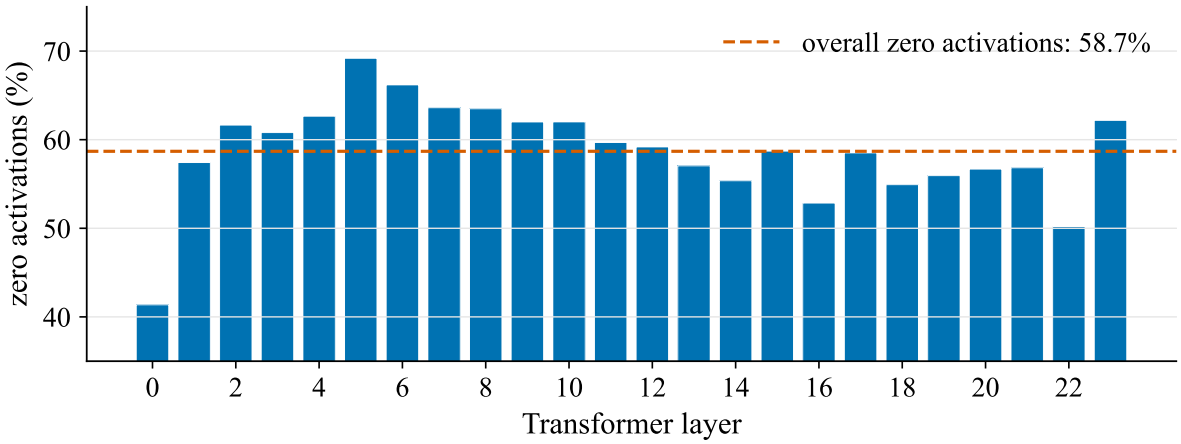


Figure 5.3: **Layerwise sparsity in the best Qwen stochastic LoRA setting.** The rank-256 FFN+attention checkpoint produces exact zeros throughout the transformer stack during a fixed gold-solution teacher-forced pass on GSM8K. The overall zero-activation rate is 58.69%, matching the sparsity value used for the memory-read estimate in Table 5.3. Layerwise zero rates range from 41.37% to 69.19%.

5.4 Language-Model-Scale Trainability

The Qwen experiment is the deployment-motivated experiment, but it leaves open a basic question: is the stochastic activation itself trainable at language-model scale? A LoRA adapter might compensate for a fragile activation on a narrow downstream task without showing that the activation can support general language-model training. To test this, two 707M-parameter GPT-style models [122] are trained from scratch on FineWeb-Edu [123]: one with deterministic SiLU and one with stochastic SiLU plus the straight-through estimator.

Both models use the same architecture and training budget. The model has 12 transformer layers, width 2048, 32 attention heads, head dimension 64, FFN dimension 8192, tied embeddings, sequence length 512, and 707,008,512 trainable parameters. The deterministic model uses ordinary SiLU in the FFN. The stochastic model replaces each FFN activation by xB with $B \sim \text{Bernoulli}(\sigma(x))$ and uses the SiLU straight-through surrogate for backpropagation. Both models are trained for approximately 2B tokens.

The purpose of this experiment is not to show that stochastic SiLU improves language-model quality. Instead, it tests stability, trainability, and sparsity. The final checkpoints are evaluated on 25M-token validation and test datasets. For the stochastic checkpoint, a posthoc mask probe repeatedly samples stochastic gates on held-out test tokens and measures the empirical on/off fraction. Detailed data, model, training, evaluation, and mask-probe settings are given in Appendix A.3.3.

Table 5.2 shows that stochastic SiLU remains trainable over the full 2B-token run, but it pays a mild performance penalty relative to the deterministic SiLU baseline. On the test set, the stochastic model has CE 3.212 compared with 3.067 for deterministic SiLU. This corresponds to test perplexity 24.817 versus 21.480, a 15.54% increase. Test top-1 next-token accuracy decreases from 41.50% to 39.79%, a gap of 1.71 percentage points. Figure 5.4 shows the same qualitative behavior during training: the stochastic model is stable, but converges to a higher validation loss.

The sparsity result is strong. The trained stochastic checkpoint has a measured off fraction

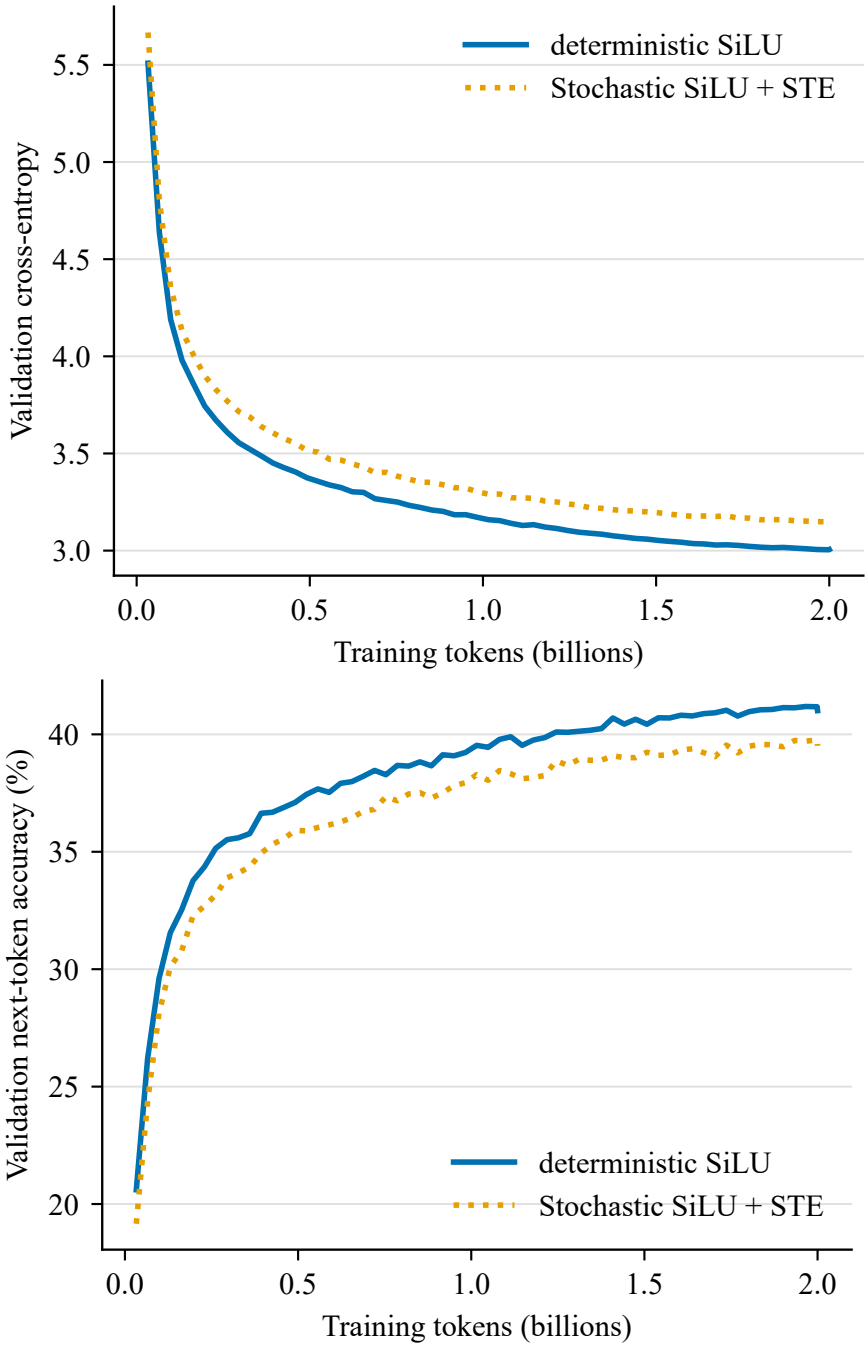


Figure 5.4: **From-scratch pretraining remains stable under stochastic SiLU.** A 707M-parameter GPT-style model trained with stochastic SiLU + STE remains stable over 2B tokens, although it converges to higher validation CE and lower next-token accuracy than the deterministic SiLU baseline.

Table 5.2: **707M-parameter GPT-style pretraining on FineWeb-Edu.** Both models use 12 layers, width 2048, FFN dimension 8192, sequence length 512, tied embeddings, and 2.000B training tokens. CE, PPL, and top-1 next-token prediction accuracy are computed on 25M-token validation and test sets from the final checkpoints.

Activation	Val CE	Val PPL	Val top-1	Test CE	Test PPL	Test top-1
Deterministic SiLU	3.068	21.505	41.47%	3.067	21.480	41.50%
Stochastic SiLU + STE	3.211	24.801	39.79%	3.212	24.817	39.79%

Activation	Zero activations
Deterministic SiLU	–
Stochastic SiLU + STE	73.58%

of 73.58% in the posthoc mask probe. This means that nearly three quarters of the stochastic FFN activation entries are exactly zero. Combined with the fact that the model trains without numerical instability over 2B tokens, this supports the broader feasibility claim: stochastic SiLU is not merely a small-model curiosity or a LoRA-only trick. It can be used throughout a moderately large language model, although the current training recipe leaves a clear quality gap relative to deterministic SiLU. This experiment is therefore a trainability check for the activation at LLM scale; the intended deployment path remains the pretrained-model LoRA retrofit in Table 5.1.

5.5 Memory-Read Opportunity from Stochastic Zeros

The two preceding experiments establish that stochastic gates can produce exact zeros in both a pretrained-model retrofit and a from-scratch training setting. The purpose of these zeros is not merely to change the mathematical form of the FFN, but to expose weight reads that can be skipped. We therefore estimate the conservative memory-read opportunity implied by the measured zero fractions.

Let ρ_{off} be the fraction of intermediate FFN channels that are exactly zero. In a two-matrix SiLU FFN, the first projection remains dense and only the second projection is treated

as skippable. If the two FFN matrices have equal size, the FFN-only read opportunity is approximately

$$\text{saving}_{\text{FFN,2mat}} \approx \frac{\rho_{\text{off}}}{2}. \tag{5.9}$$

In a SwiGLU FFN, the gate, up, and down projections are all comparable in size, and only the down projection is treated as skippable. The analogous FFN-only read opportunity is therefore approximately

$$\text{saving}_{\text{FFN,SwiGLU}} \approx \frac{\rho_{\text{off}}}{3}. \tag{5.10}$$

For a whole-model estimate, the total skippable final-projection weights are counted and divided by a dense-read denominator:

$$\text{saving}_{\text{total}} \approx \rho_{\text{off}} \frac{|W_{\text{skip}}|}{|W_{\text{dense}}|}. \tag{5.11}$$

Here $|W_{\text{skip}}|$ is the number of final-projection weights that are eligible to be skipped, and $|W_{\text{dense}}|$ is the counted dense-read denominator for the model. This estimate is intentionally a memory-read opportunity model rather than a latency model. It assumes that zero channels can be represented and consumed efficiently by a future sparse FFN runtime, and it does not attempt to model the remaining work in the transformer block.

Table 5.3 applies this model to the two language-model experiments. For Qwen2.5-0.5B, the counted linear-weight inventory is 493.96M weights, of which 104.60M are FFN down-projection weights. With the measured 58.69% zero rate from the best stochastic LoRA setting, this corresponds to a 19.6% FFN read opportunity and a 12.4% total counted-linear-read opportunity. For the 707M GPT-style model, the two-matrix FFN structure and higher measured zero rate of 73.58% imply a 36.79% FFN read opportunity and a 20.95% total parameter-read opportunity. Qwen's zero-activation rate is from the same fixed gold-solution teacher-forced pass used in Table 5.1; the 707M GPT-style value is from a posthoc mask probe on the trained stochastic checkpoint.

These estimates are deliberately optimistic about the memory path but conservative about

Table 5.3: **Conservative theoretical memory-read opportunity from exact stochastic zeros.** The projections that produce the mask remain dense; only the final FFN projection is counted as skippable when an intermediate channel is zero. These are weight-read opportunities, not end-to-end latency predictions.

Model / setting	FFN architecture	Dense-read denominator	Skippable final-projection weights
Qwen2.5-0.5B stochastic LoRA	SwiGLU: gate/up/down projections	493.96M linear weights	104.60M down-projection weights
707M GPT-style stochastic pretraining	Two-matrix SiLU FFN	707.01M total parameters	201.33M second-projection weights

Model / setting	Skippable share	Zero activations	FFN read opportunity	Total read opportunity
Qwen2.5-0.5B stochastic LoRA	21.17%	58.69%	19.6%	12.4%
707M GPT-style stochastic pretraining	28.48%	73.58%	36.79%	20.95%

what is skipped. They assume the implementation can avoid reading down-projection weights associated with zero channels, but they do not assume any savings in the projections that produce the stochastic gates. The estimates therefore answer a carefully scoped question: if stochastic activations create exact zeros at the measured rates, how much final-projection weight traffic is exposed as skippable?

5.6 Sparse FFN Kernel Opportunity

The memory-read estimates in Table 5.3 only matter if an implementation can exploit the zeros. We therefore use an isolated FFN benchmark to test whether the stochastic zeros can reduce FFN runtime when the down projection is implemented sparsely.

The benchmark uses real trained checkpoint weights from the 707M parameter GPT-style stochastic model and evaluates one-token FFN calls representative of autoregressive decoding. Two stochastic implementations are compared. The dense stochastic path evaluates

the stochastic activation but consumes the resulting hidden vector with an ordinary dense down projection. The sparse stochastic path uses the same stochastic activation, but consumes only the nonzero hidden channels in the down projection. The reported number sums the measured per-token time across all 12 FFN layers (and excludes attention layers, output projection, etc.). A concise runtime protocol is given in Appendix A.3.4.

Table 5.4 shows that the sparse FFN path reduces the measured 12-layer FFN time from 1.502066 ms/token to 1.147353 ms/token. This is a 1.309× speedup over the dense stochastic FFN implementation, or a 23.6% reduction in FFN time. The result supports the systems motivation of the chapter: once exact zeros are present, the FFN down projection can benefit from consuming only the active channels.

This result should be interpreted as an isolated FFN-level proof of opportunity, not as a completed end-to-end serving result. Full-model inference must integrate the sparse representation with attention, normalization, residual paths, output projection, and the surrounding runtime. Furthermore, the stochastic activations themselves must be implemented efficiently, which may require nontrivial engineering. In the current implementation, those integration costs are still large enough that the FFN-level gain does not become a full-model speedup relative to the end-to-end baseline using deterministic SiLU activations.

Table 5.4: **FFN-level sparse down-projection benchmark.** For the measured 12-FFN path, consuming the stochastic hidden vector sparsely reduces time from 1.502066 to 1.147353 ms/token, a 1.309× speedup relative to the dense stochastic FFN implementation. The timed path includes the up projection and stochastic activation in both cases; the dense-versus-sparse difference is how the zero-containing hidden vector is consumed by the down projection. This is an FFN-level result only, not an end-to-end model speedup.

Implementation	Time per token for 12 FFNs (ms/token)	Relative to dense stochastic FFN
Dense stochastic FFN implementation	1.502066	1.000×
Sparse stochastic FFN implementation	1.147353	1.309× faster
Difference	0.354713	23.6% lower FFN time

5.7 Discussion

Taken together, the experiments support a cautious but coherent story. The stochastic activation is mathematically simple, exactly sparse, and locally expectation-matched to SiLU. In both the Qwen retrofit and the 707M pretraining experiment, the measured zero fractions are large enough to imply nontrivial FFN down-projection memory-read opportunities under a conservative model. The isolated sparse FFN benchmark further shows that these zeros can be converted into wall-clock FFN speedup when the sparse down projection is measured directly.

The strongest practical evidence is the Qwen2.5-0.5B LoRA experiment. It shows that a pretrained model can be adapted to tolerate stochastic SwiGLU gates on a downstream reasoning task without changing the frozen base weights. The fact that FFN+attention adapters recover more effectively than FFN-only adapters suggests that stochastic gates change more than the local FFN computation; they also alter the representations passed between blocks, so attention projections benefit from adaptation as well.

The from-scratch 707M experiment provides a different kind of evidence. It does not show quality parity, but it does show stability. The stochastic model trains for the full 2B-token budget without numerical failure and produces high activation sparsity. This result is useful because it rules out the simplest negative interpretation of the LoRA result: that stochastic SiLU is fundamentally untrainable at language-model scale. Instead, the evidence suggests that stochastic SiLU is trainable and can achieve stable loss curves, but pays a mild language modeling quality tradeoff for high sparsity.

The main unresolved challenge is systems integration. The sparse FFN benchmark shows that the induced zeros can be useful when the down projection is measured directly, but a full language-model implementation must make those zeros cheap to represent and consume. The savings from skipped down-projection weight reads must exceed the overhead of carrying sparsity through the transformer block. A promising direction is therefore co-design: the activation, adaptation recipe, and sparse FFN implementation should be designed together so that zeros are produced and consumed within a low-overhead inference path.

5.8 Limitations

The limitations follow directly from the scope of the study. The benchmark measures the isolated FFN path, including the up projection, stochastic activation, and dense or sparse down projection; it does not measure full-model inference, which additionally requires attention, normalization, residual paths, output projection, and runtime integration. A completed serving result would require an integrated sparse-FFN path whose overhead is small relative to the skipped weight reads.

The Qwen2.5-0.5B experiment also has important limitations. The strongest result uses rank-256 LoRA applied to both FFN and attention projections, corresponding to 140.77M trainable adapter parameters. This is a relatively large adapter.

Generated-answer exact match and teacher-forced CE/PPL also tell different stories. The best stochastic Qwen setting recovers base-level GSM8K exact match, but its gold-solution CE and PPL remain substantially worse than the deterministic base. This implies that the adapter recovers task-level answer accuracy under the evaluation protocol, not the full deterministic token distribution.

The 707M pretraining experiment should likewise be interpreted carefully. It demonstrates stable training and high sparsity over 2B tokens, but the stochastic model remains behind the deterministic SiLU baseline in validation/test CE, perplexity, and top-1 next-token accuracy. Additional training recipes or architecture changes may reduce this gap, but they are not established by the current experiment.

Finally, the memory-read estimates are theoretical opportunities, not measured latency reductions. They assume that down-projection weight reads associated with zero channels can be skipped efficiently. They are useful for estimating the size of the opportunity, but the sparse FFN benchmark and future integrated implementations are needed to determine how much of that opportunity becomes real latency improvement.

5.9 Conclusion

Expectation-matched stochastic SiLU/SwiGLU gates offer a direct way to convert smooth dense FFN activations into exact sparse activations. The replacement is local and simple: sample a Bernoulli gate with probability $\sigma(x)$ and output xB. This preserves SiLU in pointwise expectation while creating exact zeros that expose FFN down-projection weight reads as skippable.

The experiments in this chapter show that the idea is plausible but not yet complete. A Qwen2.5-0.5B LoRA retrofit recovers base-level GSM8K generated-answer exact match under stochastic inference while maintaining approximately 59% zeros. A 707M-parameter GPT-style model trains stably with stochastic SiLU for 2B tokens and reaches approximately 74% activation zeros, although with a mild quality gap relative to deterministic SiLU. An isolated sparse FFN benchmark converts the induced zeros into a 1.309× FFN-level speedup over the dense stochastic path.

While stochastic activations in FFNs provide a promising opportunity for exploiting activation sparsity and skipping down-projection weight reads, end-to-end LLM inference efficiency remains as an engineering challenge. This chapter is best read as a forward-looking study to motivate further hardware-algorithm co-design: closing this loop requires stronger quality recovery and integrated sparse inference paths.

Chapter 6

Conclusion and Outlook

6.1 Summary of Contributions

This dissertation developed stochastic sampling as a hardware-aware computing principle in two primary settings: chaotic-bit Ising machines for optimization and sampling, and SANTA for memory-bound long-context transformer inference. A third, forward-looking study considered stochastic feedforward activations as a possible FFN-side extension. Across these settings, the object being sampled changes: Chapter 2 samples trajectories through a 2^N bitstring state space; Chapters 3–4 sample value-cache rows to reduce memory traffic in long-context attention; and Chapter 5 samples FFN channel gates to expose exact zero activations and skippable down-projection reads. The common idea is that useful computation often does not require touching every state, row, or channel at every step.

Chapter 2 evaluated and improved deterministic chaotic-bit networks that combine billiard dynamics with latches. Using similar parameters for p-bit and c-bit versions of simulated annealing and adaptive parallel tempering enabled a fair comparison of algorithmic scaling performance. Augmenting deterministic c-bits with stochasticity through probabilistic adaptive parallel tempering replica swaps gave algorithmic scaling comparable to adaptive parallel tempering using fully stochastic networks of p-bits. Beyond optimization, Chapter 2 showed

that c-bits approximately sample from the quantum Boltzmann law in a one-dimensional transverse-field Ising model and qualitatively reproduce critical-dynamics behavior in 2D Ising and 3D spin-glass systems.

Chapters 3 and 4 provided the main LLM-systems contribution of the dissertation. SANTA samples $S \ll n_k$ value rows from the post-softmax attention distribution and aggregates them with gather-and-add. The estimator is unbiased, admits practical variance reduction through stratified and systematic sampling, and provides a tunable accuracy–latency knob through the sample budget S. On Llama-3.1-8B-Instruct, the S^2ANTA kernels matched baseline accuracy on 32k-token RULER tasks and LongBench v2 while achieving up to $1.5\times$ decode-step attention-kernel speedup and up to $1.25\times$ end-to-end decode-latency speedup on RTX 6000 Ada. These results show that stochastic sampling can move beyond probabilistic search and become an operator-level tool for reducing memory traffic in long-context inference.

Chapter 5 explored stochastic neurons for sparse transformer feedforward networks as a more exploratory dissertation component. The chapter replaced deterministic SiLU/SwiGLU gates with an expectation-matched stochastic activation that samples $B \sim \text{Bernoulli}(\sigma(x))$ and outputs xB. A Qwen2.5-0.5B GSM8K LoRA retrofit recovered base-level generated-answer exact match under stochastic inference while maintaining 58.69% exact zeros. A complementary 707M-parameter GPT-style pretraining experiment showed that stochastic SiLU + STE trains stably for 2B FineWeb-Edu tokens and reaches 73.58% activation zeros, although with worse CE/PPL than deterministic SiLU. Finally, an isolated sparse FFN microbenchmark converted these zeros into a 1.309× FFN-level speedup over the dense stochastic FFN path. This result identifies an FFN-level opportunity, not a completed end-to-end serving result.

6.2 Future Directions

For chaotic-bit systems, the most important next step is a concrete hardware-aware evaluation. Such an evaluation should compare c-bits and p-bits under matched optimization and sampling

tasks while accounting for local-field update latency, latch design, topology density, energy, area, and the cost of explicit or implicit randomness. The results in Chapter 2 also suggest that deterministic chaotic dynamics and stochastic replica swaps can be combined effectively, so adaptive parallel tempering remains a useful direction for c-bits, oscillator Ising machines, and related nonlinear devices.

For SANTA, the most important future direction is integration with the broader long-context inference stack. SANTA modifies the softmax–V aggregation over whatever candidate KV state is presented to the operator, so it is naturally complementary to methods that reduce, compress, retain, offload, or select the candidate KV cache before attention is evaluated. Future work should evaluate such hybrids empirically, especially as architectures such as Multi-Head Latent Attention style models change the relative share of KV bandwidth. A second direction is adaptive sample allocation across layers, heads, contexts, and tasks, so that the sample budget is spent where attention approximation matters most.

A related systems direction is hardware/software co-design for sparse, stochastic, and adder-centric inference. The current GPU kernels primarily exploit sparse value-cache access, but SANTA also replaces dense multiply-accumulate in the value stage with gather-and-add. Hardware architectures optimized for sparse memory access and additive accumulation could therefore realize benefits beyond those measured in the GPU implementation. Bernoulli qK^{T} sampling can be pursued as a complementary score-stage path.

For stochastic feedforward activations, the next step is to close the loop between activation design, adaptation recipe, and fused sparse kernels. On the modeling side, smaller adapters, stronger adaptation recipes, and broader downstream evaluations would clarify how much quality recovery is genuinely attributable to stochastic activation adaptation. On the systems side, the sparse down-projection path must be fused tightly enough that the cost of sampling and sparse metadata does not overwhelm the skipped weight reads. This FFN direction is best viewed as a promising extension of the dissertation's sampling principle, while SANTA remains the primary demonstrated LLM inference acceleration result.

Across these directions, the central open problem is the same: identify where dense deterministic computation is wasteful, introduce stochasticity at a level where it preserves the behavior that matters, and build hardware-aware implementations that turn the resulting sparsity into real savings. The dissertation's contributions support this principle in three forms—state-space sampling for Ising machines, row sampling for attention, and channel-gate sampling for FFNs—while leaving substantial opportunities for tighter co-design between stochastic algorithms and the hardware that executes them.

Appendix A

Additional Derivations and Experimental Details

A.1 Additional Details for Chaotic Bits

Numerical simulations use double precision (64-bit) in the C++ programming language. Random numbers are generated using the Mersenne Twister engine. Spins, and initial billiard states x_i for c-bits, are randomly initialized at the beginning of every SA and APT simulation. For c-bits, we employ Euler's method with a step of $\Delta t = 0.1$ unless otherwise stated. At a given step, if the billiard x_i is greater than or equal to $+1$ or less than or equal to -1, then m_i is latched to that value.

For the 2D ferromagnetic Ising model, we obtain the critical energy experimentally. Using p-bits, we conduct 10^6 Monte Carlo sweeps at constant critical temperature $T_c = 2/\ln(1+\sqrt{2})$. We record the average energy over the last 10^5 sweeps. We take an ensemble average over 40 random seeds.

For the 3D spin glass instances, using p-bits, we conduct 10^6 Monte Carlo sweeps at constant critical temperature $T_c = 1.1$, a value found from a previous study [63]. For each of 300 problem instances, we record the average energy over the last 10^5 sweeps.

A.1.1 Kibble–Zurek Scaling Derivation and Measurement Procedure

This appendix subsection describes the details of the Kibble–Zurek Mechanism (KZM) results shown in Fig. 2.4. Although the derivation closely follows the arguments presented in Ref. [59], the purpose here is to provide precise details about the measurement procedure used in this dissertation.

According to the scaling hypothesis in phase transitions, macroscopic quantities such as relaxation times and correlation lengths diverge at the critical point. These quantities typically exhibit a power-law relationship with respect to a tuning parameter, λ. In the c-bit critical-dynamics measurements considered here, λ is the temperature, T, or its inverse, β. The power-law dependence of key quantities with respect to λ around the critical point can usually be justified by notions of scale-invariance and self-similarity:

$$\xi(\lambda) = \frac{\xi_0}{|\lambda_c - \lambda|^{\nu}} \tag{A.1}$$

where ξ is the correlation length, ν is the critical exponent for correlation lengths, and

$$\tau(\lambda) = \frac{\tau_0}{|\lambda_c - \lambda|^{z\nu}} \tag{A.2}$$

where τ is the relaxation time and $z\nu$ is the critical exponent for the relaxation time. Consider now a general annealing schedule, $\lambda(t)$, starting from an initial value $\lambda(t = 0) = \lambda_i$ and ending at the critical value $\lambda(t = t_a) = \lambda_c$. Since the key quantities of interest diverge near the critical point, the system remains roughly in equilibrium until it approaches the critical point. The non-equilibrium dynamics of interest occur close to the critical point, thus any annealing schedule $\lambda(t)$ can be linearized around λ_c, which is reached at $t = t_a$:

$$\lambda(t) = \lambda_c + \alpha(t - t_a) + O[(t - t_a)^2] \tag{A.3}$$

where α is a key quantity representing the velocity of the anneal as the system approaches the

critical point:

$$\alpha = \left.\frac{d\lambda(t)}{dt}\right|_{t=t_a} \tag{A.4}$$

As the system is annealed towards the critical point, the equilibration time increases. If the initial λ is far away from the critical point, the system quickly relaxes to its equilibrium and can follow the annealing schedule in quasi-equilibrium or adiabatically.

There comes a point, $\hat{t}$, when the remaining annealing time, $t_a - \hat{t}$, is precisely equal to the relaxation time of the system, $\tau[\lambda(\hat{t})]$. At this point, any further change in λ that increases the relaxation time results in the system not being able to equilibrate by the end of the annealing. The idea of KZM, then, is to find this "freeze-out" boundary $\hat{t}$ and to relate non-equilibrium correlation lengths to equilibrium correlation lengths via $\xi[\lambda(\hat{t})]$, which is then related to a density of topological defects via dimensional analysis. Based on this sketch, $\hat{t}$ can be found by equating $|t_a - \hat{t}|$ to $\tau(\hat{\lambda})$ using Eq. A.2 and Eq. A.3. This gives:

$$\hat{t'} = \left(\tau_0 \alpha^{-z\nu}\right)^{\frac{1}{1+z\nu}} \tag{A.5}$$

where $\hat{t'} = t_a - \hat{t}$. We then find $\lambda(\hat{t'})$ as:

$$\lambda(\hat{t'}) = \lambda_c(1 - \tau'_0 \alpha^{\frac{1}{1+z\nu}}) \tag{A.6}$$

where τ'_0 is a modified τ_0 with unimportant constants that do not affect the exponents. We then obtain the correlation lengths at freeze-out from:

$$\xi[\lambda(\hat{t})] = \epsilon'_0 \, \alpha^{-\frac{\nu}{1+z\nu}} \tag{A.7}$$

where ϵ'_0 is a modified ϵ_0 with unimportant constants. Finally, ξ is related to the density of topological defects through dimensional analysis of defect dimension d and system dimension

D:

$$\frac{\xi^d}{\xi^D} = \frac{1}{\epsilon_0'^{(D-d)}} \alpha^{\left(\frac{\nu}{1+z\nu}\right)(D-d)} \tag{A.8}$$

Eq. A.8 predicts a power-law relationship for the number of "defects" as a function of quench "velocity", α, in terms of equilibrium critical exponents. Defects in the context of Ising models are then relatable to measurable quantities such as kinks [61] and to the residual energy with similar power-law scaling laws. The critical-scaling measurements reported in this dissertation do not attempt to reproduce critical exponents for c-bits or p-bits, but instead observe the power-law dependence of c-bit and p-bit residual-energy measurements with respect to α in 2D ferro-Ising and 3D spin glass models.

A.1.2 Adaptive Parallel Tempering Preprocessing

Algorithm 2: Adaptive Parallel Tempering Preprocessing with p-bits

Input: energy variance tolerance, number of chains, number of sweeps, number of samples, constant parameter α

Output: problem-specific β schedule for parallel tempering

$i \leftarrow 1$

$\beta_1 \leftarrow 0.5$

Initialize chains to random spins

while energy variance σ_E is greater than tolerance **do**

for each chain in parallel **do**

Simulate p-bits using Eq. 2.1 and 2.2 for the specified number of Monte Carlo sweeps at constant inverse temperature β_i

Calculate variance σ_E of energies sampled at the end of the simulation

Save the spin state $\{m\}$ of each chain as the initial condition for the next iteration

Calculate the mean energy variance across chains, $\langle \sigma_E \rangle$

$\beta_{i+1} \leftarrow \beta_i + \frac{\alpha}{\langle \sigma_E \rangle}$

$i \leftarrow i + 1$

The APT preprocessing experiments use the following parameters: 100 parallel chains, 10^4 Monte Carlo sweeps, 1000 samples taken at the end of 10^4 sweeps, and $\alpha = 1.25$. The energy variance tolerance is equal to half of the smallest magnitude weight J_{ij}, and the initial value of β is set to 0.5. This preprocessing method is similar to the method used in previous work [66].

A.1.3 Time-to-Solution Metric for Factorization

To derive the time-to-solution performance metric used for the factorization benchmarks (Eq. 2.19), we start from:

$$[1 - p(Nt_a)]^k = 1 - 0.99 \tag{A.9}$$

where $p(Nt_a)$ is the probability of finding the solution using N system replicas each running for time t_a. The left-hand side of the equation represents the probability of failing k times in a row, and the right-hand side is the complement of the desired success probability, 0.99. We solve for k through algebraic manipulation. The time-to-solution is then:

$$Nt_a \times k \tag{A.10}$$

which yields the definition of time-to-solution (Eq. 2.19). This can be interpreted as the computational time per attempt, Nt_a, multiplied by the repetition count required to achieve a 99% probability of at least one successful solve.

A.2 Additional Details for SANTA

This section first gives the formal estimator proofs and sampling properties, then collects extended accuracy results, layer-sensitivity and adaptive-budget ablations, kernel pseudocode, evaluation protocols, and additional fidelity analyses that support Chapters 3–4.

A.2.1 Formal Definition of SANTA

For query q, the SANTA estimator can be written as

$$\widehat{AV}_q = \frac{1}{S} \sum_{s=1}^{S} V_{i_{q,s}}, \qquad i_{q,s} \sim \text{Categorical}(A_q). \tag{A.11}$$

Definition A.2.1 (SANTA estimator)**.** *Let*

$$Q \in \mathbb{R}^{n_q \times d_k}, \qquad K \in \mathbb{R}^{n_k \times d_k}, \qquad V \in \mathbb{R}^{n_k \times d_k},$$

and

$$A = \mathrm{softmax}\left(QK^\top/\sqrt{d_k}\right) \in \mathbb{R}^{n_q \times n_k}.$$

For each query index $r \in \{1, \dots, n_q\}$*, sample* $i_{r,1}, \dots, i_{r,S} \overset{i.i.d.}{\sim} \mathrm{Categorical}(A_r)$ *and define* $\widehat{AV}_r = \frac{1}{S}\sum_{s=1}^{S} V_{i_{r,s}}$*. Stacking all queries gives* $\widehat{AV} \in \mathbb{R}^{n_q \times d_k}$.

A.2.2 Unbiasedness and Variance Proofs

Unbiasedness of SANTA

Proposition A.2.2 (Unbiasedness)**.**

$$\mathbb{E}[\widehat{AV}] = AV.$$

Proof. Fix any q. Since $\Pr(i_{q,s} = j) = A_{qj}$,

$$\mathbb{E}[\widehat{AV}_q] = \frac{1}{S}\sum_{s=1}^{S}\sum_{j=1}^{n_k} A_{qj}V_j = \sum_{j=1}^{n_k} A_{qj}V_j = (AV)_q.$$

Linearity of expectation completes the result. □

Variance of SANTA

Throughout this subsection, fix a query index $q \in \{1, \dots, n_q\}$. Let $A_q \in \Delta^{n_k - 1}$ denote the attention weights over keys, write $p_{qj} \triangleq A_{qj}$, and let

$$\mu_q \triangleq \sum_{j=1}^{n_k} p_{qj}V_j = (AV)_q$$

be the exact attention output for that query. Default SANTA draws

$$i_{q,1}, \ldots, i_{q,S} \overset{\text{i.i.d.}}{\sim} \text{Categorical}(A_q)$$

and returns

$$\widehat{AV}_q = \frac{1}{S} \sum_{s=1}^{S} V_{i_{q,s}} \in \mathbb{R}^{d_k}.$$

Define centered summands

$$X_s \triangleq V_{i_{q,s}} - \mu_q$$

and the query-specific covariance

$$\Sigma_q \triangleq \text{Cov}(V_i) = \sum_{j=1}^{n_k} p_{qj} (V_j - \mu_q)(V_j - \mu_q)^\top.$$

Proposition A.2.3 (Variance scaling of SANTA)**.** *For default SANTA,*

$$\mathbb{E}[\widehat{AV}_q] = \mu_q, \qquad \text{Cov}(\widehat{AV}_q) = \frac{1}{S}\Sigma_q.$$

Equivalently,

$$\mathbb{E}\left[\|\widehat{AV}_q - \mu_q\|_2^2\right] = \frac{1}{S}\,\text{tr}(\Sigma_q).$$

Proof. Unbiasedness was shown in Proposition A.2.2. For the variance, write

$$\begin{aligned}
\text{Cov}\left(\widehat{AV}_q - \mu_q\right) &= \text{Cov}\left(\frac{1}{S}\sum_{s=1}^{S} X_s\right) \\
&= \frac{1}{S^2}\sum_{s=1}^{S} \text{Cov}(X_s) \\
&= \frac{1}{S}\Sigma_q.
\end{aligned}$$

Taking the trace gives the mean-squared error identity. □

A.2.3 Formal Properties of Variance-Reduced Sampling

S^2ANTA Estimator

Proposition A.2.4 (Unbiasedness of S^2ANTA)**.** *Let* $\mu_q := \sum_j p_{qj} V_j = (AV)_q$*. Then* $\mathbb{E}[\widehat{AV}_q^{\text{ind}}] = \mathbb{E}[\widehat{AV}_q^{\text{sys}}] = \mu_q$*.*

The proof is given below.

Remark A.2.5 (Systematic vs. independent)**.** *S^2ANTA-strat is unbiased and admits a variance-dominance theorem, guaranteeing lower variance than that of i.i.d. sampling with default SANTA (Section A.2.3). S^2ANTA-sys is also unbiased, but does not provide tractable variance guarantees; empirically it performs comparably.*

Unbiasedness of S^2ANTA

Proof of Proposition A.2.4. For S^2ANTA-strat,

$$\begin{aligned}
\mathbb{E}[\widehat{AV}_q^{\text{ind}}] &= \frac{1}{S} \sum_{m=0}^{S-1} \mathbb{E}\left[V_{F_q^{-1}(T_m)} \right] \\
&= \int_0^1 \left(\sum_j \mathbf{1}\{F_q(j-1) \le t < F_q(j)\} V_j \right) dt \\
&= \sum_j p_{qj} V_j \\
&= \mu_q.
\end{aligned}$$

For S^2ANTA-sys, observe that *marginally* each T_m is uniform on I_m (though dependent across m). By linearity of expectation, the sum of expectations is unchanged by this dependence, so the same integral argument applies; see [96, Ch. 8] and [97]. □

Variance Reduction for S^2ANTA-Strat

Theorem A.2.6 (Variance reduction for S^2ANTA-strat)**.** *Let* $\Sigma_q := \sum_j p_{qj} (V_j - \mu_q)(V_j -$

$\mu_q)^\top$ *and define within-stratum covariances*

$$\Sigma_q^{(m)} := \mathrm{Cov}\Big(V_{F_q^{-1}(T)} \,\Big|\, T \sim \mathrm{Unif}(I_m)\Big).$$

Then

$$\mathrm{Cov}\big(\widehat{AV}_q^{\mathrm{ind}}\big) = \frac{1}{S^2}\sum_{m=0}^{S-1} \Sigma_q^{(m)} \;\preceq_{\mathrm{Loewner}}\; \frac{1}{S}\,\Sigma_q = \mathrm{Cov}(\widehat{AV}_q),$$

with strict improvement unless all stratum means coincide. See [98, Ch. 3] and [96, Ch. 5].

Proof. Independence across strata gives

$$\mathrm{Cov}(\widehat{AV}_q^{\mathrm{ind}}) = \frac{1}{S^2}\sum_m \mathrm{Cov}\Big(V_{F_q^{-1}(T)} \mid T \in I_m\Big).$$

By the law of total covariance with $T \sim \mathrm{Unif}(0,1)$ and partition $\{I_m\}$,

$$\begin{aligned}\Sigma_q &= \mathbb{E}\Big[\mathrm{Cov}\Big(V_{F_q^{-1}(T)} \mid T \in I_m\Big)\Big] \\ &\quad + \mathrm{Cov}\Big(\mathbb{E}\Big[V_{F_q^{-1}(T)} \mid T \in I_m\Big]\Big) \\ &\succeq_{\mathrm{Loewner}} \mathbb{E}\Big[\mathrm{Cov}\Big(V_{F_q^{-1}(T)} \mid T \in I_m\Big)\Big].\end{aligned}$$

and averaging then dividing by S yields the claim. Strictness holds unless the stratum means are all equal. □

Corollary A.2.7 (MSE ordering)**.** *For any q,*

$$\mathbb{E}\Big[\|\widehat{AV}_q^{\mathrm{ind}} - \mu_q\|_2^2\Big] \le \mathbb{E}\big[\|\widehat{AV}_q - \mu_q\|_2^2\big] = \frac{1}{S}\,\mathrm{tr}(\Sigma_q).$$

A.2.4 Concentration Bound for SANTA

Theorem A.2.8 (Vector Bernstein tail for SANTA)**.**

Assume that, for all j,

$$\|V_j - \mu_q\|_2 \le V_{\max}, \qquad \textit{and hence} \qquad \|X_s\|_2 \le V_{\max}.$$

Let

$$\nu_q \triangleq \mathbb{E}\left[\|V_i - \mu_q\|_2^2\right] = \operatorname{tr}(\Sigma_q).$$

Then, for all $t > 0$,

$$\Pr\left(\left\|\widehat{AV}_q - \mu_q\right\|_2 \ge t\right) \le 2\exp\left(-\frac{S\,t^2}{2\,(\nu_q + V_{\max} t/3)}\right).$$

Equivalently, with probability at least $1 - \delta$,

$$\left\|\widehat{AV}_q - \mu_q\right\|_2 \le \sqrt{\frac{2\,\nu_q\,\log(2/\delta)}{S}} + \frac{2V_{\max}\log(2/\delta)}{3S}.$$

Proof. Fix q and the attention inputs, and let $X_s = V_{i_{q,s}} - \mu_q$. The X_s are independent mean-zero vectors in $\mathbb{R}^{d_k}$ with $\|X_s\|_2 \le V_{\max}$ and $\sum_{s=1}^{S} \mathbb{E}\|X_s\|_2^2 = S\nu_q$. If $\nu_q = 0$, then $X_s = 0$ almost surely and the conclusion is immediate. Otherwise, their partial sums form a Hilbert-space martingale. Apply [95, Thm. 3.4] with $D = 1$, $a = V_{\max}$, $b^2 = S\nu_q$, and threshold $u > 0$. Using

$$h(z) := (1+z)\log(1+z) - z \ge \frac{z^2}{2(1+z/3)}, \qquad z \ge 0,$$

the resulting Bennett bound implies

$$\Pr\left(\left\|\sum_{s=1}^{S} X_s\right\|_2 \ge u\right) \le 2\exp\left(-\frac{u^2}{2(S\nu_q + V_{\max}u/3)}\right).$$

Set $u = St$ to obtain the stated tail bound. For $0 < \delta < 1$, substituting

$$t = \sqrt{\frac{2v_q \log(2/\delta)}{S}} + \frac{2V_{\max} \log(2/\delta)}{3S}$$

makes the right-hand side at most δ, proving the high-probability statement. □

A.2.5 Bernoulli $qK^\top$ Derivations

We first normalize the query vector as $q \leftarrow q/\text{norm}$, where $\text{norm} \geq \max_i |q_i|$ so that $|q_i| \in [0, 1]$ can be interpreted as a Bernoulli probability. Then, we introduce the unbiased query estimator $\hat{q}$ where each element $\widehat{q_i}$ is an independent ternary Bernoulli variable:

$$\begin{aligned} b_i &\sim \text{Bernoulli}(|q_i|), \\ \widehat{q_i} &= b_i \times \text{sign}(q_i) \in \{-1, 0, +1\}. \end{aligned} \tag{A.12}$$

Sampling from $\hat{q}$ provides an unbiased estimator for the product $p = qK^\top$:

$$\widehat{p} = \frac{\text{norm}}{B} \sum_{n=1}^{B} \hat{q}^{(n)} K^\top, \tag{A.13}$$

where $\hat{q}^{(n)}$ is a Bernoulli query vector, and B is the number of samples.

Bernoulli $qK^\top$ Error

Figure A.1 shows the scaling of the L2 norm error with the number of samples B, defined as $e = \|\widehat{p} - p\|_2 / \|p\|_2$. In similar fashion as SANTA, stratified Bernoulli sampling reduces the variance of each independent element $\widehat{p_i}$. The variance scales as $O(1/B^2)$, resulting in an overall L2 norm scaling as $O(1/B)$. For this numerical experiment, which uses a head dimension $d_k = 128$ and a context length $n_k = 1024$, using only $B = 4$ samples induces a mean L2 norm error of approximately 60% for standard Bernoulli sampling, compared to a much lower 30% for stratified sampling.

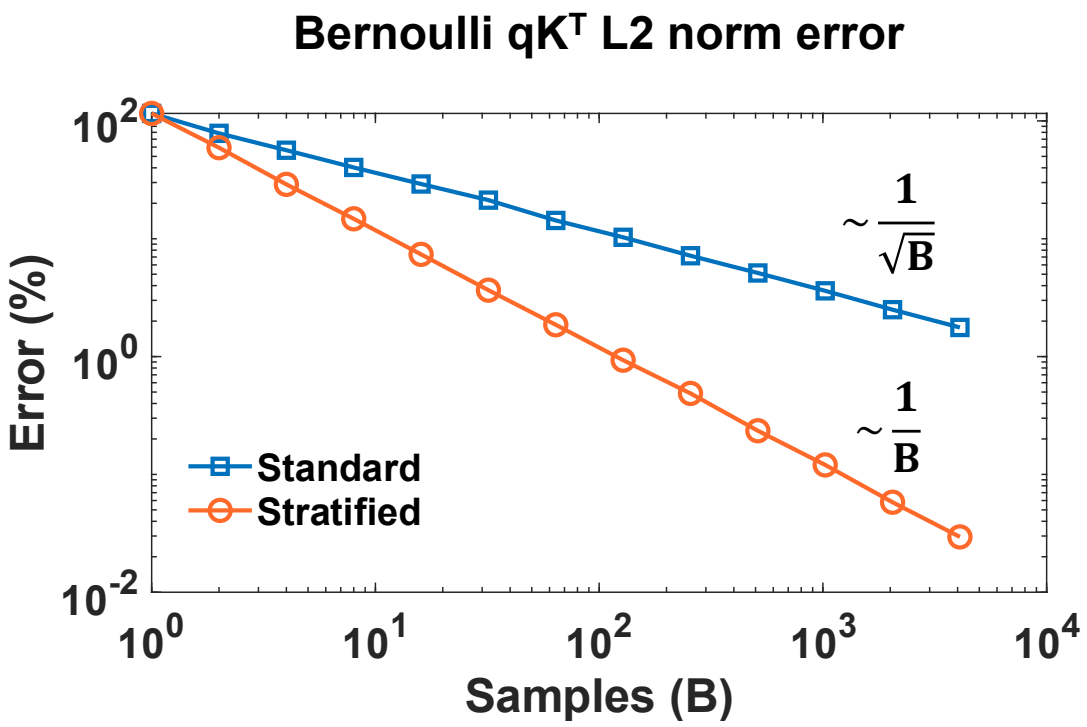


Figure A.1: Mean L2 norm error for Bernoulli $qK^\top$. The score error is calculated during decoding with $d_k = 128$, a context length $n_k = 1024$, and averaged over 100 instances $q \sim \mathcal{N}(0,1)$, $K \sim \mathcal{N}(0,1)/\sqrt{d}_k$, $V \sim \mathcal{N}(0,1)$.

$qK^\top$ for Grouped Query Attention

To maintain sparse key access when keys share multiple queries, as in grouped-query attention (GQA), we define a common representative query for the group, such as the mean of the absolute values of the queries:

$$m = \frac{1}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} |q_g|, \tag{A.14}$$

where $\mathcal{G}$ denotes the query group. We then estimate the mean query using Monte Carlo sampling, after normalizing by norm $\geq \max_i m_i$:

$$b_i \sim \text{Bernoulli}(m_i/\text{norm}) \tag{A.15}$$

$$\widehat{m} = \frac{\text{norm}}{B} \sum_{n=1}^{B} b^{(n)} \tag{A.16}$$

where $b^{(n)}$ are Bernoulli query vectors with independent elements b_i. After fetching the corresponding K^T rows corresponding to non-zero elements of $\widehat{m}$, the product $p = qK^\mathsf{T}$ for each query is estimated as:

$$\widehat{p} = \left(\widehat{m} \odot \frac{q}{m}\right) K^\mathsf{T}, \tag{A.17}$$

where $\odot$ denotes element-wise multiplication. In this mean group query scenario, calculating $\widehat{p}$ requires multiplications by the original query q and normalization by m to compensate for the bias introduced by the mean query approximation.

A.2.6 Empirical Variance and Unique-Key Measurements

Empirical variance. To further validate our results, we empirically measure the variance of SANTA, S^2ANTA-strat, and S^2ANTA-sys. We sweep the sample budget S on 8k-token single-hop QA prompts, averaged across model layers and 30 prompts (normalized per head). S^2ANTA variants exhibit significantly lower variance than SANTA. Though systematic sampling admits no tractable theoretical variance guarantees, it empirically demonstrates similar variance to independent stratified sampling (and requires 1 random number per query instead of S random numbers). On a log-log scale, slopes for SANTA, S^2ANTA-strat, and S^2ANTA-sys are -1.1, -1.27, and -1.29, respectively, matching the $1/S$ scaling prescribed by theory; the measurement protocol is described below.

Unique-key count. Since all versions of SANTA sample keys with replacement, the number of *unique* keys may be less than the sample budget S. In Table A.1, we empirically measure the

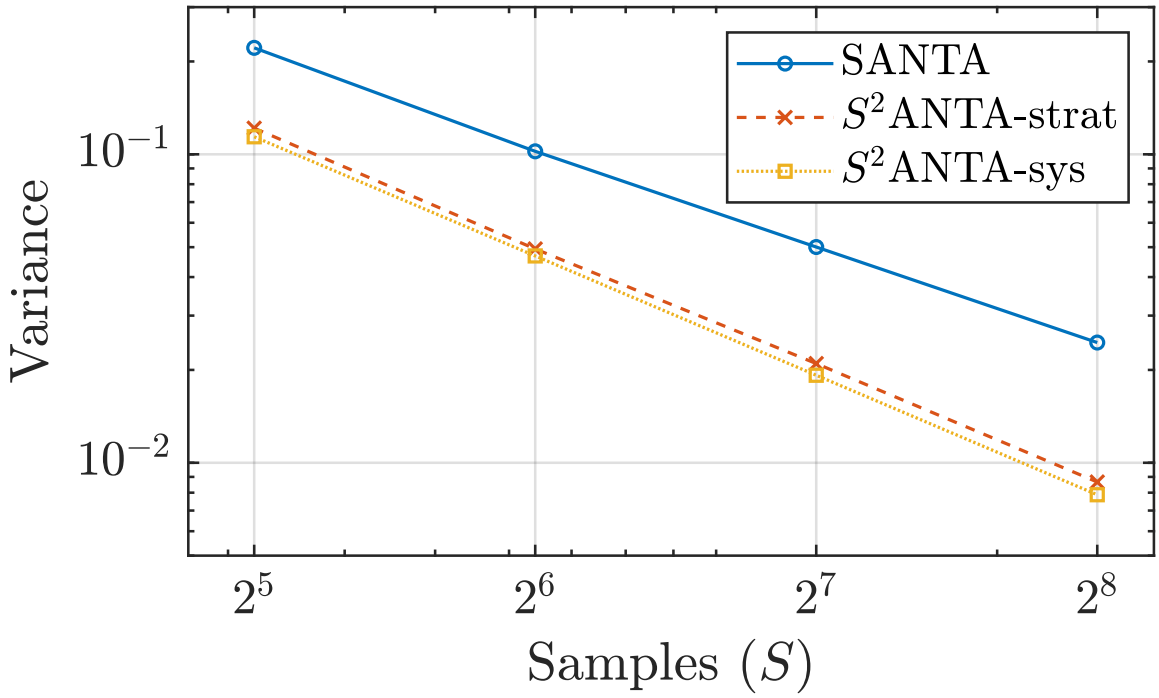


Figure A.2: Empirical variance of SANTA on single-hop QA prompts (8k tokens).

Table A.1: Unique-key count vs. S (8192 tokens, single-query).

S	SANTA	S^2ANTA-strat	S^2ANTA-sys
32	11.0	12.7	13.0
64	16.7	21.2	21.7
128	25.9	35.8	36.4
256	38.3	60.3	61.6

average number of unique keys sampled by the last query position for 8192-token single-hop QA prompts (normalized per head and averaged across model layers). This is indicative of the key diversity observed in autoregressive decoding. For a given S, S^2ANTA variants achieve higher key diversity than default SANTA. Furthermore, the number of unique keys tends to be $\ll S$, implying that *fewer than S* unique rows of V require memory accesses. The measurement protocol is described below.

Implementation and measurement protocol. For a single layer/head at the last prefill position $q^\star$, let the post-softmax attention over n_k keys be $p_j \geq 0$ with $\sum_{j=1}^{n_k} p_j = 1$, and let

$V_j \in \mathbb{R}^{d_k}$ denote the corresponding value vectors. The dense-attention mean is

$$\mu = \sum_{j=1}^{n_k} p_j V_j \in \mathbb{R}^{d_k}.$$

SANTA estimator. Given a per-head sampling budget S, all SANTA variants estimate μ with

$$\widehat{AV} = \frac{1}{S} \sum_{s=1}^{S} V_{J_s},$$

where the key indices $\{J_s\}$ are drawn by one of three schemes: (i) multinomial; (ii) independent equal-mass stratified; or (iii) systematic (random-start, fixed-stride). These are the same variants used in the SANTA chapters; the code paths that implement them also record the per-head statistics described below at $q^\star$.

What we measure, per head at $q^\star$. We record the following two quantities.

(1) Unique keys. The number of distinct keys touched by the sampler,

$$U := \left|\{J_1, \ldots, J_S\}\right|,$$

upper-bounded by S (can be $< S$ if high-probability keys repeat). This is a proxy for how many memory locations each head actually reads.

(2) Variance trace of the estimator. We report the trace of the covariance (expected squared ℓ_2 error) of $\widehat{AV}$,

$$\mathrm{VarTrace}(\widehat{AV}) := \mathbb{E}\left[\|\widehat{AV} - \mu\|_2^2\right] = \mathrm{tr}\left(\mathrm{Cov}(\widehat{AV})\right),$$

specialized to each sampling scheme:

- **Multinomial (closed form).** Let $\Sigma := \sum_{j=1}^{n_k} p_j (V_j - \mu)(V_j - \mu)^\top$ so that $\mathrm{tr}(\Sigma) = \sum_j p_j \|V_j\|_2^2 - \|\mu\|_2^2$. Then

$$\mathrm{VarTrace}_{\mathrm{multi}}(\widehat{AV}) \;=\; \frac{1}{S}\,\mathrm{tr}(\Sigma).$$

- **Independent equal-mass stratified (closed form).** Partition $[0, 1)$ into S equal strata; for each stratum m, form the within-stratum discrete distribution induced by p and let Σ_m be the corresponding value covariance. Because draws across strata are independent,

$$\mathrm{VarTrace}_{\mathrm{strat}}(\widehat{AV}) \;=\; \frac{1}{S^2}\sum_{m=0}^{S-1}\mathrm{tr}(\Sigma_m).$$

- **Systematic (replicate-based estimate).** There is no general closed form that holds uniformly for systematic sampling, so we estimate variance by repeated random starts. With $R \geq 2$ independent offsets, we compute $\widehat{AV}^{(r)}$ for each replicate, then use

$$\widehat{\mathrm{VarTrace}}_{\mathrm{sys}}(\widehat{AV}) = \frac{1}{R-1}\sum_{r=1}^{R}\|\widehat{AV}^{(r)} - \overline{V}\|_2^2,$$
$$\overline{V} = \frac{1}{R}\sum_{r=1}^{R}\widehat{AV}^{(r)}.$$

  This estimator is applied per head at $q^\star$.

How we aggregate and report. For each prompt, we compute U and the variance-trace scalar per head at $q^\star$, then average over heads within a layer, average over layers, and finally average over prompts to obtain the quantities reported in the variance plots:

$$\overline{U}(S, \text{variant}) \quad \text{and} \quad \overline{T}(S, \text{variant}) \;=\; \overline{\mathrm{VarTrace}}(\widehat{AV}).$$

This yields the curves summarized above as a function of the budget S and the SANTA variant.

Protocol summary. All measurements use evaluation-mode forward passes, prefill-only, and record statistics *at the last token* per head; no additional dense attention pass is performed to form empirical errors. For multinomial and stratified we use the exact formulas above; for systematic we use the replicate-based estimator.

A.2.7 Extended Accuracy Results and Top-k Comparisons

Table A.2: GSM8K accuracy and average context length (prompt + answer) for DeepSeek-R1-Distill-Qwen-7B. k: number of keys in top-k, S: SANTA sample budget. Accuracy shows 95% bootstrap confidence intervals.

	Top-k		SANTA	
k|S	Acc. (%)	Tok.	Acc. (%)	Tok.
2	0.23±0.15	4049	0.00±0.00	4217
4	23.88±1.31	3298	0.08±0.09	4208
8	72.25±1.35	2332	52.46±1.57	2799
16	82.51±1.21	1860	79.00±1.28	1778
32	86.23±1.02	1694	83.19±1.23	1667
64	86.76±1.03	1624	84.99±1.09	1587
128	87.54±1.02	1626	85.42±1.00	1594
256	88.20±0.97	1600	86.93±1.01	1627
SDPA (baseline)			**88.75±1.00**	1532

Table A.3: GSM8K accuracy and average context length (prompt + answer) for Llama-3.1-8B-Instruct.

	Top-k		SANTA		S^2ANTA-strat		S^2ANTA-sys	
k|S	Acc. (%)	Tok.	Acc. (%)	Tok.	Acc. (%)	Tok.	Acc. (%)	Tok.
2	1.21±0.32	430	1.26±0.34	1075	1.01±0.32	1071	1.11±0.34	1071
4	6.27±0.72	583	1.57±0.37	825	1.54±0.38	611	1.67±0.40	569
8	41.32±1.50	549	1.44±0.38	207	1.74±0.40	325	2.10±0.45	340
16	62.88±1.57	430	5.51±0.72	350	39.12±1.50	352	44.63±1.58	349
32	72.50±1.34	384	38.26±1.43	348	67.00±1.48	343	68.59±1.42	339
64	76.12±1.32	358	63.63±1.49	346	74.43±1.31	343	**76.42±1.34**	341
128	77.13±1.30	349	70.23±1.42	341	75.64±1.33	341	**77.33±1.34**	342
256	78.19±1.33	340	75.61±1.42	342	**78.17±1.28**	343	**77.56±1.34**	343
SDPA (baseline)							**78.06±1.33**	344

Table A.4: MMLU accuracy and average context length (prompt + answer) for DeepSeek-R1-Distill-Qwen-7B. k: number of keys in top-k, S: SANTA sample budget. Accuracy shows 95% bootstrap confidence intervals.

	Top-k		SANTA	
k|S	Acc. (%)	Tok.	Acc. (%)	Tok.
2	24.97±1.24	3598	4.33±0.58	4178
4	38.93±1.32	2236	23.10±1.20	3666
8	55.08±1.47	1659	44.55±1.47	2080
16	59.07±1.36	1301	58.81±1.42	1506
32	60.77±1.40	1173	61.05±1.45	1228
64	62.03±1.38	1103	60.59±1.43	1104
128	61.68±1.49	1061	**62.38±1.32**	1075
256	62.05±1.46	1041	**62.27±1.39**	1074
SDPA (baseline)			**63.16±1.45**	1020

Table A.5: MMLU accuracy and average context length (prompt + answer) for Llama-3.1-8B-Instruct.

	Top-k		SANTA		S^2ANTA-strat		S^2ANTA-sys	
k|S	Acc. (%)	Tok.	Acc. (%)	Tok.	Acc. (%)	Tok.	Acc. (%)	Tok.
2	23.56±1.20	588	24.89±1.27	1144	24.28±1.21	1141	24.52±1.22	1141
4	28.37±1.27	359	25.13±1.20	961	24.65±1.26	807	24.49±1.18	774
8	39.54±1.43	532	25.06±1.23	284	25.63±1.23	286	24.47±1.23	292
16	45.26±1.37	484	26.24±1.26	313	32.46±1.32	350	34.14±1.39	353
32	49.49±1.44	448	33.81±1.34	354	43.78±1.50	398	45.48±1.45	403
64	48.33±1.37	433	41.11±1.40	397	**49.25±1.48**	415	**48.70±1.39**	418
128	49.47±1.44	418	47.46±1.45	412	**49.92±1.47**	421	**50.60±1.37**	421
256	49.75±1.43	421	**49.75±1.51**	417	**50.90±1.49**	422	**51.12±1.46**	421
SDPA (baseline)							**49.86±1.47**	424

Table A.6: Accuracy of Llama 8B on 4 long context tasks with an 8k-token prompt. Tasks: frequent word extraction (FWE), needle-in-a-haystack (NIAH), single-hop QA (QA_1), multi-hop QA (QA_2). k: number of keys for top-k. S: SANTA sample budget. Accuracy shows 95% bootstrap CIs.

Kernel	k|S	FWE	NIAH	QA_1	QA_2
S^2ANTA-sys	64	96.27±0.97	94.05±1.07	71.80±3.90	67.20±4.20
S^2ANTA-sys	128	97.80±0.70	94.15±1.02	68.20±4.10	66.80±4.00
S^2ANTA-sys	**256**	**97.87±0.70**	**95.00±0.95**	**71.20±3.90**	**69.40±4.10**
S^2ANTA-strat	64	95.93±0.94	94.35±1.15	71.40±4.00	65.40±4.20
S^2ANTA-strat	128	97.73±0.80	94.30±1.10	70.80±4.20	67.40±4.20
S^2ANTA-strat	**256**	**98.33±0.60**	**94.55±1.05**	**71.20±4.00**	**67.00±4.20**
SANTA	64	92.53±1.27	93.95±1.30	64.60±4.20	63.40±4.20
SANTA	128	94.53±1.13	91.45±1.38	67.80±4.10	63.40±4.30
SANTA	256	96.20±0.93	93.15±1.18	68.00±4.10	66.40±4.10
SDPA (baseline)	**—**	**98.53±0.60**	**94.55±1.00**	**71.80±3.90**	**68.60±4.20**
top-k	64	93.07±1.20	92.55±1.10	65.60±4.10	60.80±4.10
top-k	128	95.40±0.96	93.60±1.15	69.20±3.90	62.20±4.50
top-k	**256**	**97.27±0.80**	**94.55±1.05**	**70.60±3.90**	**64.20±4.10**

Tables A.2, A.3, A.4, and A.5 feature additional GSM8K and MMLU results with comparisons using DeepSeek-R1-Distill-Qwen-7B and top-k attention. Table A.6 exhibits long-context benchmark results identical to Table 3.6, except with additional top-k comparisons. These results also show that SANTA does not dominate top-k in every regime, especially at small budgets on some reasoning benchmarks, so the comparison should be interpreted together with the operation-mix discussion below.

Comparing SANTA and top-k requires care, as they have different computational profiles. We present results for sample budget S equal to the top-k budget k not as an iso-compute benchmark, but to demonstrate that SANTA achieves comparable or superior accuracy while using a fundamentally cheaper value-stage operation mix. At $S = k$, top-k costs $n_q k d_k$ multiplications plus $n_q k d_k$ additions, whereas SANTA avoids the dense value-stage weighted multiplications, costs $n_q S d_k$ additions, and retains only the fixed $n_q d_k$ normalizing divisions or multiplies counted in the main-text FLOP/memory accounting of Table 3.3.

On modern CMOS, a 32-bit FP multiply costs 3.7 pJ versus 0.9 pJ for addition [126]. For equal budgets $k = S$, the value-stage energy consumption for the two methods is approximately $k \times (3.7 + 0.9) = 4.6k$ pJ per element for top-k, versus $S \times 0.9 = 0.9S$ pJ per element for SANTA's sampled additions, ignoring the fixed normalization term that does not scale with k or S. Relative to top-k, this gives an $\approx 5\times$ reduction in value-stage arithmetic energy for the sampled weighted-sum portion. We acknowledge that these are approximate energy estimates; the goal is to provide hardware-agnostic comparisons, as modern GPU hardware is optimized for matmuls and contiguous memory access, while SANTA involves adds and sparse memory access.

Regarding memory accesses: SANTA's accesses to elements of the V matrix scale as $n_q S d_k$, which *in the worst case* matches top-k's $n_q k d_k$ memory accesses at $k = S$. However, since SANTA samples with replacement, the number of *unique* keys is $\leq S$. In practice, as we quantify in Appendix A.2.6, the number of unique keys sampled is $< S$, which may offer significant caching advantages.

For benchmarking, we apply SANTA and top-k to both prefill and autoregressive decoding. However, in practice, these sparse attention implementations may see greater benefit during decoding, when only a single query sparsely selects rows of V.

SANTA variants demonstrate strong performance relative to top-k on GSM8K and MMLU benchmarks, despite using a similar number of additions and value accesses at $k = S$ while removing the dense value-stage weighted multiplications.

A.2.8 Extended LongBench v2 Results

Table A.7 reports the full LongBench v2 budget sweep corresponding to the compact operating-point summary in Table 4.2. Accuracy is reported with 95% bootstrap confidence intervals. Results are averaged over three runs; prompts longer than 32k tokens are truncated to 32k following the repository protocol. The main operating points, S^2ANTA-prop with S=128 and S^2ANTA-flash with S=2048, match SDPA within bootstrap confidence intervals in this 30k-token regime. Lower-budget S^2ANTA-flash settings degrade substantially, consistent with the sample-waste behavior discussed in Section 4.4, while S^2ANTA-prop remains near the SDPA baseline across the tested budgets.

Table A.7: Full LongBench v2 sweep for Llama 8B. S: sample budget. Accuracy shows 95% bootstrap confidence intervals.

Kernel	S	Acc. (%)	Avg. scored tok.
S^2ANTA-flash	128	8.748±2.964	30035.5
S^2ANTA-flash	256	14.049±3.504	30024.0
S^2ANTA-flash	512	26.574±3.740	29990.0
S^2ANTA-flash	1024	27.170±1.509	29994.2
S^2ANTA-flash	**2048**	**28.827±3.422**	**29996.4**
S^2ANTA-prop	64	29.092±2.534	29997.5
S^2ANTA-prop	**128**	**28.098±1.028**	**29998.4**
S^2ANTA-prop	256	28.694±1.029	29995.4
S^2ANTA-prop	1024	28.562±1.588	29996.2
SDPA (baseline)	**—**	**28.363±1.509**	**29996.0**

A.2.9 Layer Sensitivity and Sample-Budget Ablations

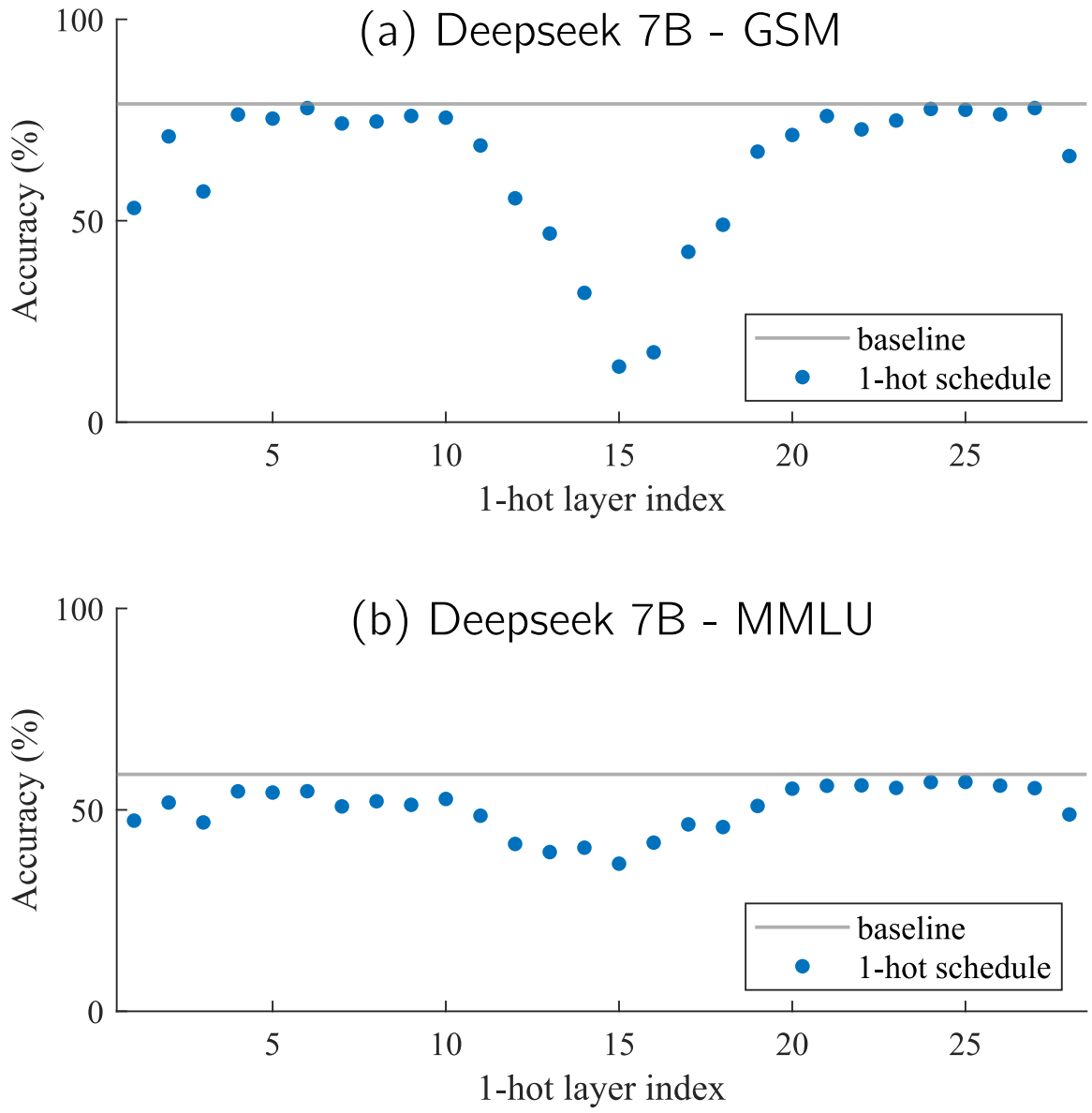


Figure A.3: Ablations with one-hot stochastic attention. Panels (a) and (b) show DeepSeek 7B results on GSM8K and MMLU. Baseline scores employ a fixed sampling budget of 16 samples per layer. The horizontal axis shows the index of the model layer which is reduced to stochastic hard attention.

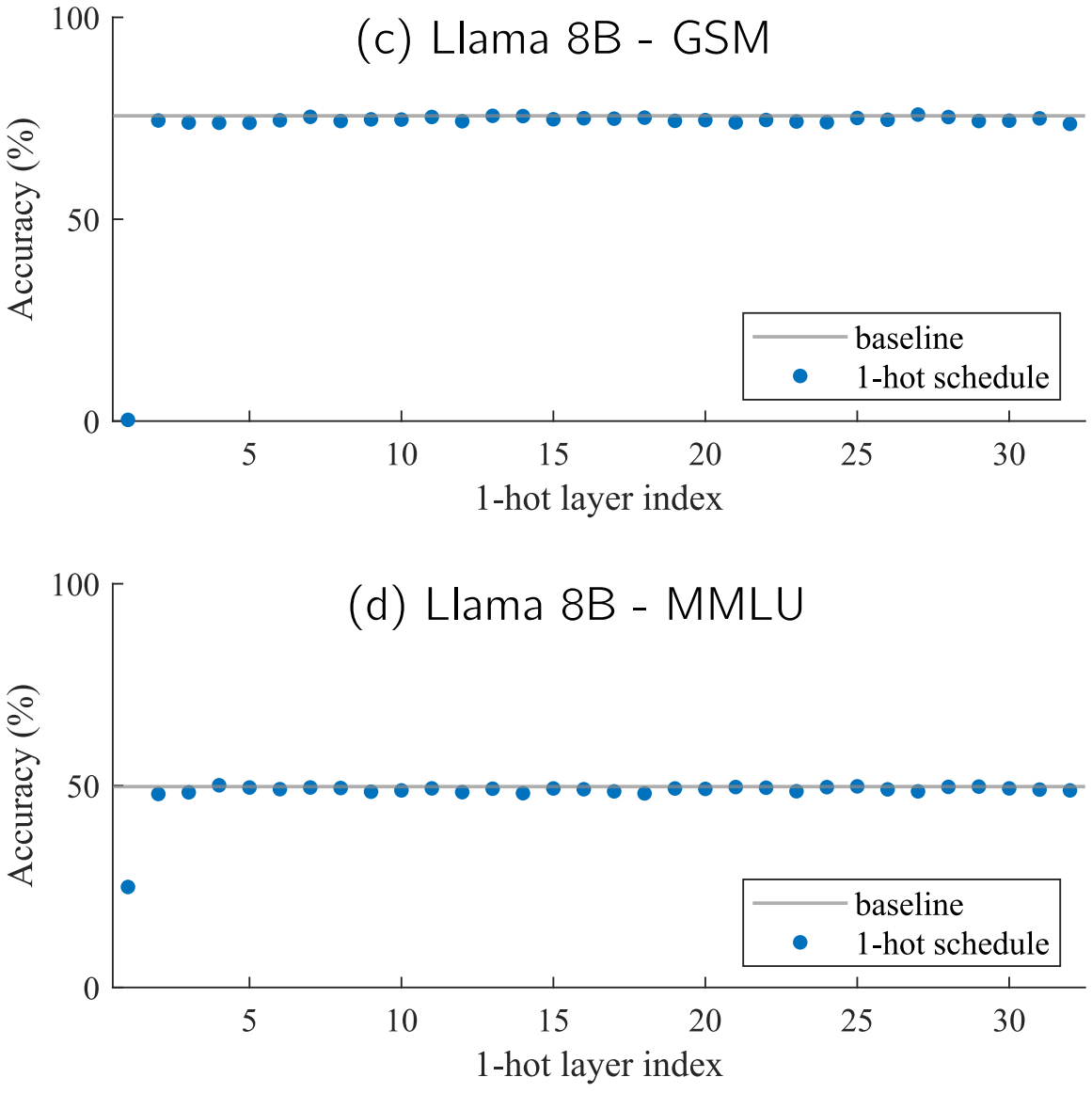


Figure A.3: Ablations with one-hot stochastic attention (continued). Panels (c) and (d) show Llama 8B results on GSM8K and MMLU, using a baseline budget of 256 samples per layer.

This appendix section probes a fundamental question: how much does each attention layer matter? We find that one-hot stochastic attention, a limiting case of SANTA with $S = 1$, acts as a surprisingly sharp diagnostic. It reveals which transformer layers are robust to severe approximation and which are not, offering a practical lens into where attention precision matters most.

We perform these ablations by setting $S = 1$ in a single layer while keeping all other layers at fixed sample budgets: $S = 16$ for DeepSeek 7B (28 layers) and $S = 256$ for Llama 8B (32 layers). In this setting, each query attends to only one randomly sampled key, making that layer a hard attention bottleneck.

Models are evaluated on GSM8K and MMLU with the same protocol as Tables 3.4 and 3.5. The effect of this localized bottleneck on accuracy is summarized in Fig. A.3. Some layers exhibit almost no degradation when ablated in this way, while others collapse entirely, most notably middle layers in DeepSeek 7B and the first layer in Llama 8B.

In DeepSeek 7B, layers 12–18 are highly sensitive to approximation, while early and final layers are much more tolerant. In contrast, Llama 8B exhibits extreme sensitivity in just the first layer. These patterns suggest that attention importance is neither uniform nor trivially architectural.

We observe similar qualitative trends regardless of the baseline sample budget; the $S = 16$ and $S = 256$ settings are arbitrary. The early-layer sensitivity in Llama 8B is consistent with prior findings on high-entropy attention in shallow layers [127], and with prior work that avoids approximating early layers [76]. The middle-layer fragility in DeepSeek 7B remains an open and intriguing observation.

Algorithm 3: Layer-wise sample-budget optimization via REINFORCE

Require: initial logits $\boldsymbol{\theta} \in \mathbb{R}^{28}$ ($\theta_i = 0$), total budget N=224, learning-rate α=0.02

while training **do**

Workers (in parallel):

for $e = 1, \ldots, E$ // E=10 *episodes / worker run* **do**

Compute $p_i \leftarrow \exp(\theta_i) / \sum_j \exp(\theta_j)$

Sample schedule $\mathbf{S} \sim \text{Multinomial}(N, \mathbf{p})$

Evaluate 16 GSM8K questions using schedule **s**; obtain reward $r \in [0, 1]$

Append $(\mathbf{s}, r)$ to shared folder

Aggregator:

Collect all new $(\mathbf{s}^{(k)}, r^{(k)})$ tuples // $k = 1, \ldots, K$

Compute baseline $b \leftarrow \frac{1}{K} \sum_k r^{(k)}$

Compute gradient $\mathbf{g} \leftarrow \frac{1}{K} \sum_k (r^{(k)} - b)\left(\mathbf{s}^{(k)} - N\mathbf{p}\right)$

Update logits $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\, \mathbf{g}$

Write back updated $\boldsymbol{\theta}$

We now consider a global sample budget shared across all 28 transformer layers of DeepSeek 7B, allowing each layer to have a distinct sample count. We optimize this allocation using a REINFORCE-style algorithm [128], detailed in Algorithm 3.

We train the policy on the GSM8K training set. The policy logits $\boldsymbol{\theta} \in \mathbb{R}^{28}$ define a softmax distribution over layers. At each iteration, we sample schedules using a multinomial draw over this distribution, with a total budget of 224 samples (i.e., 8×28). Each episode evaluates 16 questions; the reward is the fraction answered correctly. We intentionally starve the baseline with $S = 8$ samples per layer to create room for the learned schedule to improve.

Fig. A.4 shows the final learned schedule after 501 iterations. The allocation aligns strikingly with the ablation trends from Fig. A.3: more samples are assigned to the first and middle layers, where one-hot attention was most detrimental. That REINFORCE independently rediscovers this structure from only reward signals confirms that these patterns are not artifacts of the

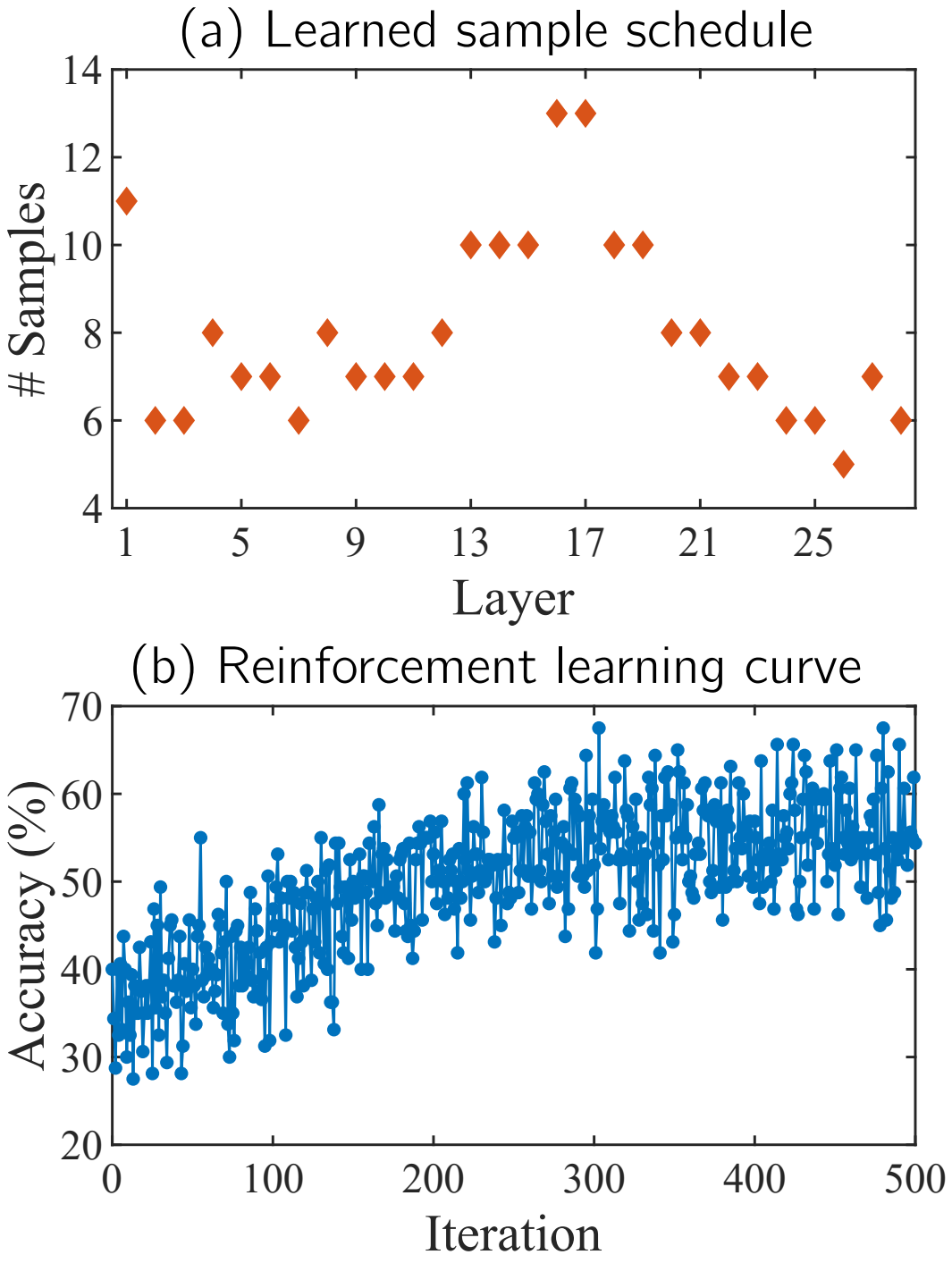


Figure A.4: (a) Learned schedule with a total budget of 224 samples across all 28 transformer blocks (DeepSeek 7B). (b) RL iterations over time.

ablation method but instead are intrinsic to the models.

This experiment is not tuned for peak accuracy, but demonstrates that adaptive per-layer budgets meaningfully outperform fixed ones.

Table A.8: RL sample-schedule performance. Accuracy shows 95% bootstrap confidence intervals.

Schedule	Task	Accuracy (%)
Baseline (8 samples)	GSM8K	52.46 ± 1.47
Baseline (8 samples)	MMLU	44.55 ± 1.47
Learned schedule	GSM8K	$\mathbf{66.06 \pm 1.47}$
Learned schedule	MMLU	$\mathbf{48.86 \pm 1.53}$

A.2.10 Distributed RL Implementation Details

The preceding subsection treats adaptive sample budgets as an exploratory accuracy ablation. This subsection records the distributed implementation used to run that ablation; it is not part of the main SANTA kernel path.

Topology. All pods mount a single read–write volume that serves as the rendezvous point.

- **Aggregator pod.** Holds the current policy and records a running history of baselines and gradients.
- **Worker pods.** Operate in a tight loop: *(i)* read the latest policy; *(ii)* sample a 28-dimensional schedule; *(iii)* evaluate ten batches of 16 questions; *(iv)* append a small JSON-lines result file to a designated "inbox" directory on the volume.

Aggregator loop. Whenever at least one new result file appears, the aggregator moves all current files into a private folder for the next iteration, computes the REINFORCE update and writes a new policy snapshot. It never pauses or signals the workers; it simply processes whatever has arrived. In practice, workers are unlikely to finish evaluation simultaneously, thus nearly every policy update comes from one worker output (10×16 questions). If two land together, the aggregator just uses a double-sized batch, which lowers the gradient's variance.

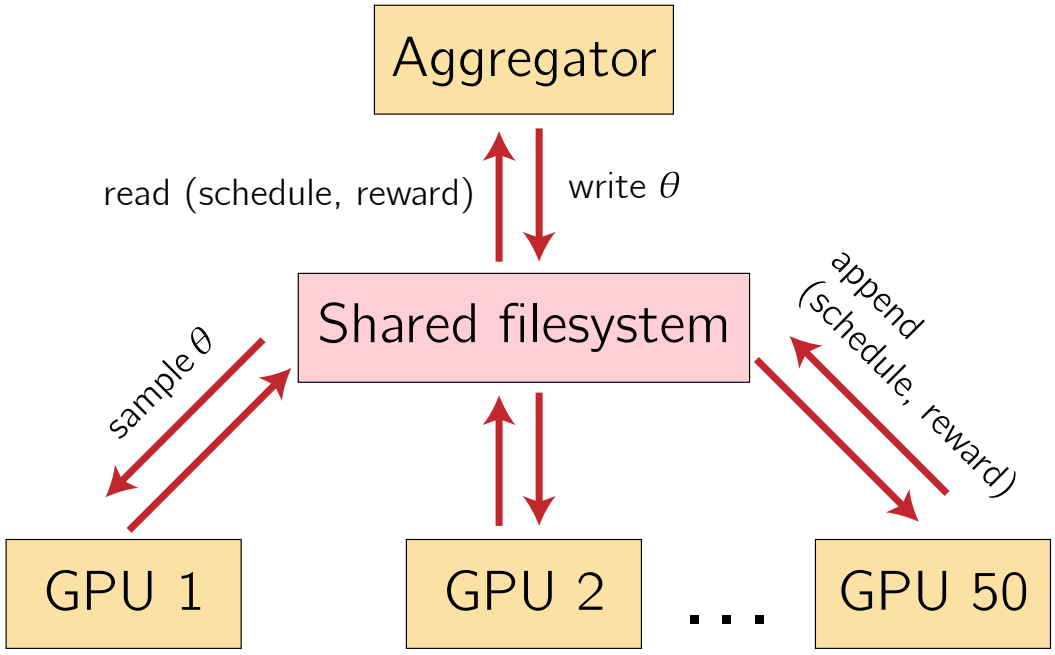


Figure A.5: Kubernetes deployment schematic.

Asynchrony and fault-tolerance.

- Workers can be terminated at any time; partial output vanishes harmlessly.
- Workers always proceed with the newest policy that has reached disk; no locking is needed.
- If the aggregator restarts it scans the history directory, resumes from the last completed iteration, and continues.

Scaling and resources. The worker Deployment's replica count can be scaled up or down at any time during learning. For the 501 iterations in Fig. A.4 we ran 50 GPU workers on a mix of RTX 3090, L4, and A10 cards (24 GB) for roughly 20 hours, totaling ≈ 1000 GPU-hours. The aggregator requests just one CPU core and a few hundred MiB of RAM.

Terminology. We keep the names aggregator and worker to highlight a simple parameter-server design. These roles align with the common learner/actor split in distributed reinforcement

learning.

A.2.11 Evaluation Protocols

For the GSM8K evaluation in Section 3.6.1, the GSM8K test split (1319 prompts) is evaluated 3 times. Evaluation code uses straightforward answer parsing and borrows grading code from PRM800K [129]. For all runs, temperature $= 0.6$, top-p $= 0.95$, and repetition penalty $= 1.1$. Left-padded batches of 8–16 prompts are used.

For the MMLU evaluation in Section 3.6.2, results are averaged over three runs on the MMLU validation set (1531 questions), using identical decoding parameters as GSM8K.

For the long-context benchmark results in Section 3.6.3, results are averaged over 500 prompts and greedy decoding is employed with no repetition penalty applied.

For the 32k-token kernel results in Section 4.4 and the LongBench v2 results in Appendix A.2.8, LongBench v2 [110] results are averaged over three runs. Prompts longer than 32k tokens are truncated to 32k following the repository protocol. Accuracy is reported with 95% bootstrap confidence intervals and the average number of scored tokens.

DeepSeek 7B + GSM8K

Table A.9: Prompting and grading for DeepSeek 7B on GSM8K.

Aspect	**Procedure**
Prompt	Question, blank line, then `Please reason step by step, and put your final numeric answer within \boxed{}.`
Answer extraction	Perform a brace-balanced search for the last occurrence of `\boxed{…}` (robust to nested braces). If none is found, fall back to the final ∼200 characters of the model output.
Grading	Numeric equality judged by the MIT-licensed PRM800K grader; every prompt is counted, and blank or unparsable predictions score 0.

DeepSeek 7B + MMLU

Table A.10: Prompting and grading for DeepSeek 7B on MMLU.

Aspect	Procedure
Prompt	Question, four labeled choices, then `Answer briefly. Put your final answer as a single letter inside \boxed{}.`
Answer extraction	Brace-balanced search for the last occurrence of `\boxed{…}`; if none is found, inspect the final ∼300 characters of the output. Scan this region backwards and take the first occurrence of A–D (case-insensitive), then convert it to uppercase.
Grading	The predicted letter is compared with the gold key (case-insensitive). If no valid letter is found—or the output is blank—the item scores 0; otherwise it scores 1 when the letters match.

Llama 8B + GSM8K

Table A.11: Prompting and grading for Llama 8B on GSM8K.

Aspect	Procedure
Prompt	Meta chat template. *System*: "You are a precise mathematician. Explain your reasoning step by step, then output ONLY the final number on a new line." *User*: the GSM8K question.
Answer extraction	Inspect the model reply in order: (i) the last non-empty line of the response; (ii) if that fails, the final 25 whitespace-separated tokens. From the selected chunk, take the first integer-looking token (commas allowed, optional leading minus).
Grading	Numeric equality judged by the MIT-licensed PRM800K grader; every question is counted, and blank or unparsable answers score 0.

Llama 8B + MMLU

Table A.12: Prompting and grading for Llama 8B on MMLU.

Aspect	Procedure
Prompt	Meta chat template. *System*: "You are a knowledgeable and concise subject-matter expert. Work through the problem step by step. Finally, on a new line, output ONLY the single capital letter (A, B, C, or D) that corresponds to the correct choice." *User*: the question followed by the four labeled choices A–D.
Answer extraction	Take the substring after the last newline and scan it left-to-right for the first capital A–D (case-insensitive). If none is found, inspect the final 10 whitespace-separated tokens of the whole response, scanning them right-to-left for A–D.
Grading	Predicted letter (upper-cased) is compared with the gold key. Every question is counted; if no valid letter is found or it does not match the gold answer, the item scores 0.

Llama 8B + RULER prompts

Table A.13: Prompting and grading for Llama 8B on RULER prompts.

Aspect	Procedure
Prompt	Prompts are generated for the following RULER sub-tasks with a length of 4096–32768 tokens: fwe, niah_multivalue, qa_1, and qa_2. Greedy decoding is used. The model generates a maximum of 64 new tokens.
Grading	fwe: partial credit = (# of gold words present in the model output) / 3 (case-insensitive set match). niah_multivalue: take the first four unique integers from the model output; partial credit = (# of these that appear in the gold set) / 4. qa_1 and qa_2: correct if the gold answer string appears as a case-insensitive substring of the model output.

Licenses.

- **Models.** deepseek-ai/DeepSeek-R1-Distill-Qwen-7B (MIT) and meta-llama/Meta-Llama-3.1-8B-Instruct (Llama 3.1 license) on Hugging Face.
- **Datasets.** GSM8K openai/gsm8k (MIT) and MMLU cais/mmlu (MIT) on Hugging Face. RULER prompts (Apache).
- **Grader.** PRM800K numeric grader (MIT).

A.2.12 S^2ANTA-Prop Pseudocode

Algorithm 4 provides a high-level overview of S^2ANTA-prop. Algorithms 5, 6, and 7 show breakdowns of the kernels that perform score computation, sample budget allocation, and the V gather, respectively.

Algorithm 4: S^{2}ANTA-prop (overview, decode/GQA): proportional tile budgets + systematic sparse-V

Input: $Q \in \mathbb{R}^{H \times d_k}$; $K, V \in \mathbb{R}^{n_k \times H_{\text{kv}} \times d_k}$ (GQA); tile length B_{tile}; samples/head S.

Output: $O \in \mathbb{R}^{H \times d_k}$ approximating $\text{softmax}(QK^\top)V$.

$T \leftarrow \lceil n_k / B_{\text{tile}} \rceil, \quad G \leftarrow H / H_{\text{kv}}, \quad k(h) \leftarrow \lfloor h/G \rfloor, \quad \mathcal{T}_t = [t B_{\text{tile}}, \min((t+1)B_{\text{tile}}, n_k))$

Pass 1: compute $m_{h,t} = \max_{n \in \mathcal{T}_t} s_{h,n}$ and $\ell_{h,t} = \sum_{n \in \mathcal{T}_t} \exp(s_{h,n} - m_{h,t})$; optionally stash $u_{h,n} = \exp(s_{h,n} - m_{h,t})$, where $s_{h,n} = Q_h^\top K_{n,k(h)} / \sqrt{d_k}$

Budgets: $m_h^\star = \max_t m_{h,t}; \quad W_{h,t} = \exp(m_{h,t} - m_h^\star)\, \ell_{h,t}; \quad$ allocate integers $\{S_{h,t}\}_t$ with $\sum_t S_{h,t} = S$ and $S_{h,t} \approx S \cdot W_{h,t} / \sum_t W_{h,t}$ (largest remainder); set $\text{inv}\delta_{h,t} = S_{h,t} / \ell_{h,t}$ and $a_{0,h,t} \sim U[0,1)$

Pass 2: systematic counts $c_{h,n}$ from $x_{h,n} = \text{inv}\delta_{h,t} u_{h,n}$, then $O_h \leftarrow O_h + \sum_n c_{h,n} V_{n,k(h)}$

return O/S

Algorithm 5: S^{2}ANTA-prop, Kernel 1: pass-1 tile statistics (and optional u-stash)

Input: $Q \in \mathbb{R}^{H \times d_k}$; $K \in \mathbb{R}^{n_k \times H_{\text{kv}} \times d_k}$; tile length B_{tile}.

Output: $m \in \mathbb{R}^{H \times T}$, $\ell \in \mathbb{R}^{H \times T}$, and optional $u \in \mathbb{R}^{H \times n_k}$.

$T \leftarrow \lceil n_k / B_{\text{tile}} \rceil; \quad G \leftarrow H / H_{\text{kv}}$

for each KV head $k \in \{0, \ldots, H_{\text{kv}} - 1\}$ and tile $t \in \{0, \ldots, T-1\}$ **in parallel do**

  $\mathcal{T}_t \leftarrow [t B_{\text{tile}}, \min((t+1)B_{\text{tile}}, n_k))$

  for each grouped head $g \in \{0, \ldots, G-1\}$ **do**

    $h \leftarrow kG + g; \quad s_{h,n} \leftarrow Q_h^\top K_{n,k} / \sqrt{d_k}$ for all $n \in \mathcal{T}_t$

    $m_{h,t} \leftarrow \max_{n \in \mathcal{T}_t} s_{h,n}; \quad \ell_{h,t} \leftarrow \sum_{n \in \mathcal{T}_t} \exp(s_{h,n} - m_{h,t})$

    (Optional) $u_{h,n} \leftarrow \exp(s_{h,n} - m_{h,t})$ for all $n \in \mathcal{T}_t$

Algorithm 6: S^{2}ANTA-prop, Kernel 2: proportional per-tile budgets (largest remainder)

Input: Tile stats $m_{h,t}, \ell_{h,t}$; samples/head S; seed.

Output: Integer budgets $S_{h,t}$ with $\sum_t S_{h,t} = S$, plus inv$\delta_{h,t}$ and offsets $a_{0,h,t}$.

for each head $h \in \{0, \dots, H-1\}$ **in parallel do**

$m_h^\star \leftarrow \max_t m_{h,t}$

$W_{h,t} \leftarrow \exp(m_{h,t} - m_h^\star)\, \ell_{h,t}$; $\quad Z_h \leftarrow \sum_{t=0}^{T-1} W_{h,t}$

$q_t \leftarrow S \cdot W_{h,t} / Z_h$; $\quad S_{h,t} \leftarrow \lfloor q_t \rfloor$ for all t

Give remaining $S - \sum_t S_{h,t}$ samples to tiles with largest fractional part of q_t (largest remainder)

inv$\delta_{h,t} \leftarrow (S_{h,t} > 0)\ ?\ S_{h,t}/\ell_{h,t}\ :\ 0$; $\quad a_{0,h,t} \sim \mathrm{Unif}(0,1)$

Algorithm 7: S^{2}ANTA-prop, Kernel 3: systematic resampling and sparse-V accumulation (GQA grouped)

Input: $V \in \mathbb{R}^{n_k \times H_{\text{kv}} \times d_k}$; $u_{h,n}$; inv$\delta_{h,t}, a_{0,h,t}$; tile length B_{tile}.

Output: Accumulator $O \in \mathbb{R}^{H \times d_k}$ (fp32).

$G \leftarrow H / H_{\text{kv}}$; $\quad O \leftarrow 0$

for each KV head $k \in \{0, \dots, H_{\text{kv}}-1\}$ and tile $t \in \{0, \dots, T-1\}$ **in parallel do**

$\mathcal{T}_t \leftarrow [t B_{\text{tile}}, \min((t+1) B_{\text{tile}}, n_k))$; $\quad$ heads $\mathcal{H}_k = \{kG, \dots, (k+1)G-1\}$

for each head $h \in \mathcal{H}_k$ **do**

$p \leftarrow 0$

for each $n \in \mathcal{T}_t$ (increasing) **do**

$x \leftarrow \text{inv}\delta_{h,t}\, u_{h,n}$

$c_{h,n} \leftarrow \lfloor a_{0,h,t} + p + x \rfloor - \lfloor a_{0,h,t} + p \rfloor$; $\quad p \leftarrow p + x$

(Optional) compact emitted rows $\mathcal{E}_t = \{n \in \mathcal{T}_t : \exists h \in \mathcal{H}_k,\ c_{h,n} > 0\}$

for each $n \in \mathcal{E}_t$ and each $h \in \mathcal{H}_k$ **do**

$O_h \leftarrow O_h + c_{h,n} V_{n,k}$

return O (apply final scaling by $1/S$ outside or in a final cast/scale kernel)

A.2.13 S^2ANTA-Flash Pseudocode

Algorithm 8 provides a high-level overview of S^2ANTA-flash. Algorithms 9 and 10 describe the 2 kernels that constitute S^2ANTA-flash in detail.

Algorithm 8: S^{2}ANTA-flash (overview, decode/GQA): uniform per-tile sampling + FlashDecoding-style merge

Input: $Q \in \mathbb{R}^{H \times d_k}$; $K, V \in \mathbb{R}^{n_k \times H_{\text{kv}} \times d_k}$ (GQA); tile length B_{tile}; samples/head S; seed.

Output: $O \in \mathbb{R}^{H \times d_k}$ approximating $\text{softmax}(QK^\top)V$.

$T \leftarrow \lceil n_k / B_{\text{tile}} \rceil$, $G \leftarrow H / H_{\text{kv}}$, $k(h) \leftarrow \lfloor h/G \rfloor$; uniform per-tile budget $S_{\text{tile}} \approx S/T$ (rounded).

Kernel 1 (tile partials): for each tile (k, t), compute $(m_{h,t}, \ell_{h,t})$ and $\widetilde{O}_{h,t} = \sum_{n \in \mathcal{T}_t} c_{h,n} V_{n,k}$ via systematic sampling with budget S_{tile}.

Kernel 2 (merge): for each head h, set $m_h^\star = \max_t m_{h,t}$, $W_{h,t} = \exp(m_{h,t} - m_h^\star)\ell_{h,t}$, $Z_h = \sum_t W_{h,t}$, and $O_h = \frac{1}{Z_h} \sum_t W_{h,t} \left(\widetilde{O}_{h,t} / S_{\text{tile}}\right)$.

return O.

Algorithm 9: S^{2}ANTA-flash, Kernel 1: fused per-tile partials (stats + systematic sampling)

Input: $Q \in \mathbb{R}^{H \times d_k}$; $K, V \in \mathbb{R}^{n_k \times H_{\text{kv}} \times d_k}$; tile length B_{tile}; samples/head S; seed.

Output: Per-tile outputs $(m_{h,t}, \ell_{h,t}, \widetilde{O}_{h,t})$ for all heads and tiles.

$T \leftarrow \lceil n_k / B_{\text{tile}} \rceil$, $G \leftarrow H / H_{\text{kv}}$, $S_{\text{tile}} \approx S/T$ (rounded).

for each KV head $k \in \{0, \dots, H_{\text{kv}} - 1\}$ and tile $t \in \{0, \dots, T-1\}$ **in parallel do**

$\mathcal{T}_t \leftarrow [t B_{\text{tile}}, \min((t+1)B_{\text{tile}}, n_k))$; heads $\mathcal{H}_k = \{kG, \dots, (k+1)G - 1\}$.

for each head $h \in \mathcal{H}_k$ **do**

$s_{h,n} \leftarrow Q_h^\top K_{n,k} / \sqrt{d_k}$ for $n \in \mathcal{T}_t$.

$m_{h,t} \leftarrow \max_{n \in \mathcal{T}_t} s_{h,n}$, $u_{h,n} \leftarrow \exp(s_{h,n} - m_{h,t})$, $\ell_{h,t} \leftarrow \sum_{n \in \mathcal{T}_t} u_{h,n}$.

$\text{inv}\delta_{h,t} \leftarrow (S_{\text{tile}} / \ell_{h,t})$ (or 0 if $\ell_{h,t}$ is tiny); $a_{0,h,t} \sim U[0, 1)$.

Use systematic counts $c_{h,n}$ from $x_{h,n} = \text{inv}\delta_{h,t} u_{h,n}$ and accumulate $\widetilde{O}_{h,t} \leftarrow \sum_{n \in \mathcal{T}_t} c_{h,n} V_{n,k}$.

Algorithm 10: S^{2}ANTA-flash, Kernel 2: FlashAttention-style merge across tiles

Input: Per-tile $(m_{h,t}, \ell_{h,t}, \widetilde{O}_{h,t})$ from Alg. 9; S_{tile}.

Output: Final $O \in \mathbb{R}^{H \times d_k}$.

for each head $h \in \{0, \dots, H-1\}$ **in parallel do**

$m_h^\star \leftarrow \max_t m_{h,t}$.

$W_{h,t} \leftarrow \exp(m_{h,t} - m_h^\star)\, \ell_{h,t}$; $Z_h \leftarrow \sum_{t=0}^{T-1} W_{h,t}$.

$O_h \leftarrow \frac{1}{Z_h} \sum_{t=0}^{T-1} W_{h,t} \left(\widetilde{O}_{h,t} / S_{\text{tile}}\right)$.

return O.

A.2.14 Decode-Step Microbenchmark Protocol

This subsection describes the microbenchmark used to measure *decode-step* attention latency for FlashAttention2 decoding ("FlashDecoding"), FlashInfer decoding, and the custom S^2ANTA CUDA kernels. The intent is to be explicit about tensor shapes, software/hardware environment, cache state, and what is (and is not) included in the reported runtimes, in order to support a fair comparison.

What is being measured. We measure GPU kernel time for a single decoding step (one new token) as a function of the KV-cache length n_k. Timing is performed with CUDA events, recorded on the active CUDA stream and synchronized on the end event each iteration. The reported milliseconds therefore reflect elapsed time on the GPU timeline for the CUDA work launched by a single backend call (which may internally launch multiple kernels). All Python/CPU overhead and input construction are excluded by allocating and populating tensors outside the timed region and recording events immediately around the backend call.

Hardware and driver. All measurements were run on a single **NVIDIA RTX 6000 Ada Generation** GPU (48 GB). The installed driver reported by `nvidia-smi` was 580.82.07 (CUDA compatibility reported as 13.0).

Software environment (versions). The benchmark environment used:

- **PyTorch 2.6.0+cu124**
- **flash-attn 2.7.1** (FlashAttention2)
- **flashinfer-python 0.5.3**

The S^2ANTA kernels are invoked through a PyTorch CUDA extension (custom CUDA code callable from PyTorch); the timed region includes all CUDA kernels launched by this extension entrypoint.

Decode geometry and tensor shapes. We benchmark a single-batch decode step with head geometry matching a Llama-style GQA configuration:

$$H = 32, \quad H_{\text{kv}} = 8, \quad d_k = 128, \quad G = H/H_{\text{kv}} = 4,$$

where H is the number of query heads, H_{kv} is the number of KV heads, and d_k is the head dimension. The query length is one token (decode), batch size is one, and attention is causal.

For each cache length n_k, the following tensors are constructed once and reused across timing iterations:

- Query tensor with shape $[H, d_k]$.
- Key/value cache tensors stored in a sequence-major layout. The tensor shape is

$$[\, n_k{+}1,\ H_{\text{kv}},\ d_k \,],$$

  where the extra $+1$ slot corresponds to the current token.

All backends are run with the same causal setting and the standard softmax scaling $1/\sqrt{d_k}$, with dropout disabled.

Input distribution and dtype. All inputs are i.i.d. Gaussian (`torch.randn`) generated directly on the GPU. Unless otherwise stated, the query, keys, and values are provided in bfloat16 (the harness also supports fp16). A fixed RNG seed is used so that, for a given n_k, each backend sees the same input tensors.

Cold-cache protocol (L2 thrashing). To reduce sensitivity to favorable cache residency effects: before *each* timed iteration, we touch a large GPU buffer by performing an in-place update on a 512 MB float32 tensor. This operation is intended to thrash GPU caches (including L2) via a large read+write stream. The cache-thrash operation is executed *outside* the timed

region (before the start event is recorded), and therefore is not included in the reported kernel time.

Timing protocol and trial structure. For each cache length n_k and each backend, we run a fixed number of warmup and timed iterations:

- **Warmup:** 10 untimed iterations to amortize one-time effects (e.g., kernel selection/caching inside libraries).
- **Timing:** 40 timed iterations. Each iteration performs an optional cache-thrash (not timed), then records a start CUDA event, launches the backend once, records an end CUDA event, synchronizes on the end event, and reads elapsed milliseconds from the event pair.
- **Aggregation:** we report the mean latency over the 40 timed iterations.

All tensor allocations and concatenations/appends used to construct inputs are performed outside the timed region.

Note on KV-cache appends in FlashDecoding. FlashAttention2 decoding performs a fused KV-cache append/update as part of its decode call, whereas FlashInfer decoding and the S^2ANTA decode kernels do not perform an internal append in this benchmark. To isolate the attention-like decode computation for FlashInfer and S^2ANTA, we provide a cache tensor that already includes the current token in the preallocated $[n_k+1, H_{\text{kv}}, d_k]$ storage.

This asymmetry is not material in the long-context regime: the fused append cost is $O(H_{\text{kv}} d_k)$ per step and does not scale with n_k, so its fractional contribution decreases as n_k grows. Speedup comparisons focus on large KV-cache lengths (e.g., 8k–32k tokens), where the attention computation dominates. Including the fused append inside FlashDecoding therefore does not change qualitative conclusions in this regime (and makes the FlashDecoding measurement slightly more conservative relative to an "attention-only" timing).

A.2.15 Additional Kernel Accuracy Results

Table A.14: Accuracy of Llama 8B on 4 long-context tasks with an 8k-token prompt. Tasks: frequent word extraction (FWE), needle-in-a-haystack (NIAH), single-hop QA (QA_1), multi-hop QA (QA_2). S: sample budget. Accuracy shows 95% bootstrap confidence intervals.

Kernel	S	FWE	NIAH	QA_1	QA_2
S^2ANTA-prop	64	96.60±0.52	93.23±0.69	70.73±2.43	67.33±2.46
S^2ANTA-prop	**128**	**98.04±0.40**	**93.78±0.60**	**70.33±2.40**	**66.87±2.43**
S^2ANTA-prop	256	98.07±0.40	94.05±0.62	70.67±2.33	66.40±2.30
S^2ANTA-flash	64	66.47±1.12	83.57±1.05	65.67±2.44	62.27±2.40
S^2ANTA-flash	128	86.11±0.91	91.70±0.73	69.47±2.30	66.27±2.50
S^2ANTA-flash	256	94.80±0.62	92.77±0.63	70.67±2.20	66.80±2.50
S^2ANTA-flash	**1024**	**98.11±0.40**	**94.22±0.58**	**70.73±2.30**	**66.33±2.40**
SDPA (baseline)	**—**	**98.53±0.60**	**94.55±1.00**	**71.80±3.90**	**68.60±4.20**

A.2.16 HumanEval Code-Generation Results

Table A.15 reports HumanEval pass@1 accuracy [130] for Llama-3.1-8B-Instruct using the S^2ANTA-prop and S^2ANTA-flash kernels. We include HumanEval as an additional benchmark-diversity check. Because HumanEval prompts are short relative to the long-context decode regime targeted by SANTA, these results are intended to characterize accuracy rather than demonstrate kernel speedups. At $S = 256$, both S^2ANTA-flash and S^2ANTA-prop remain within the bootstrap confidence interval of the SDPA baseline.

Protocol. We evaluate the standard HumanEval pass@1 setting with greedy decoding and compare model-generated completions against the unit tests for each problem. We report pass@1 with 95% bootstrap confidence intervals and the average number of generated/evaluated tokens. Since HumanEval examples are short-context code-generation tasks, they do not exercise the memory-bound 30k–32k-token decode regime emphasized in the main kernel benchmarks. Accordingly, we use HumanEval only to verify that the approximation does not catastrophically degrade code-generation accuracy at moderate budgets.

Table A.15: HumanEval pass@1 accuracy for Llama 8B. S: sample budget. Pass@1 shows 95% bootstrap confidence intervals.

Method	S	Pass@1 (%)	Avg. tok.
S^2ANTA-flash	16	12.805±4.370	521.6
S^2ANTA-flash	32	44.106±6.707	296.5
S^2ANTA-flash	64	59.350±6.504	267.7
S^2ANTA-flash	128	60.569±6.809	263.0
S^2ANTA-flash	**256**	**63.008±6.911**	**260.1**
S^2ANTA-prop	16	46.545±6.301	274.6
S^2ANTA-prop	32	61.789±6.504	267.1
S^2ANTA-prop	64	60.366±7.012	262.5
S^2ANTA-prop	128	62.398±6.911	262.5
S^2ANTA-prop	**256**	**63.211±7.012**	**261.1**
SDPA (baseline)	**—**	**65.244±7.317**	**261.4**

A.2.17 Same-State Fidelity and Tile-Sensitivity Analyses

This subsection reports additional analyses that characterize when the S^2ANTA kernels are most faithful to dense attention. We focus on 32k-token frequent-word extraction (FWE) and multi-hop QA (QA_2) prompts, since these are representative long-context tasks used in the main kernel-accuracy evaluation. Unless otherwise stated, we select 40 long prompts per task and examine 32 decoding steps per prompt. At each step, using the dense model's current prefix as the reference context, we apply the S^2ANTA approximation only in the final decoding layer and compare the resulting attention output AV against the corresponding dense-attention output at the same decode state. Relative L2 error and cosine similarity are averaged over attention heads and evaluated decoding steps. Same-state top-1 agreement is measured by running a full one-token forward pass and checking whether the approximate model preserves the dense model's top-1 next-token prediction.

Numerical Approximation Error

Table A.16 reports same-state fidelity for S^2ANTA-flash and S^2ANTA-prop. Approximation quality improves with the sample budget S. The low same-state top-1 agreement for S^2ANTA-

flash at $S = 256$ is consistent with its downstream accuracy drop in Table 4.1. By contrast, the main operating points used in Section 4.4, namely $S = 2048$ for S^2ANTA-flash and $S = 128$ for S^2ANTA-prop, recover high cosine similarity and high top-1 next-token agreement.

Table A.16: Same-state numerical fidelity of S^2ANTA-flash and S^2ANTA-prop on 32k-token FWE and QA_2 prompts. Relative L2 error and cosine similarity compare final-layer sparse and dense attention outputs AV at matched decode states. Top-1 agreement reports whether a full one-token approximate forward pass preserves the dense model's top-1 next-token prediction.

Task	Backend	S	Mean relative L2 error	Mean cosine similarity	Same-state top-1 agreement
FWE	flash	256	0.244	0.963	91%
FWE	flash	1024	0.091	0.993	97%
FWE	flash	2048	0.055	0.997	97%
FWE	prop	128	0.131	0.986	96%
FWE	prop	256	0.085	0.994	97%
FWE	prop	1024	0.035	0.999	98%
QA_2	flash	256	0.572	0.851	91%
QA_2	flash	1024	0.233	0.968	97%
QA_2	flash	2048	0.144	0.986	99%
QA_2	prop	128	0.208	0.967	99%
QA_2	prop	256	0.141	0.983	97%
QA_2	prop	1024	0.063	0.996	99%

Sensitivity of S^2ANTA-Flash to Attention Support Across Tiles

Table A.17 studies how S^2ANTA-flash fidelity changes as attention mass spreads across more tiles. We fix the S^2ANTA-flash sample budget to $S = 2048$, matching the main 32k-token operating point in Section 4.4, and use tile size 256. At each decode position, we compute the dense attention distribution and count how many tiles are needed to capture 90% of total attention mass. We then bucket observations by this tile-support statistic.

S^2ANTA-flash is most accurate when most attention mass is concentrated in a small number of tiles. Fidelity degrades as attention mass becomes more diffuse, especially on QA_2, where relevant evidence may be spread across the context. This behavior is consistent with the "sample waste" explanation in Section 4.4: uniform per-tile sampling spends samples on low-mass tiles

before deferred renormalization.

Table A.17: Sensitivity of S^2ANTA-flash to attention support spread across tiles. Tile size is fixed to 256 and sample budget is fixed to $S = 2048$. "Tiles to reach 90% attention mass" is computed from the dense attention distribution at each evaluated decode state.

Task	Tiles to reach 90% attention mass	Mean relative L2 error	Mean cosine similarity
FWE	1 tile	0.035	0.998
FWE	2 tiles	0.047	0.998
FWE	3–4 tiles	0.068	0.997
FWE	5–8 tiles	0.084	0.995
FWE	9+ tiles	0.073	0.996
QA_2	1 tile	0.055	0.997
QA_2	2 tiles	0.117	0.991
QA_2	3–4 tiles	0.170	0.982
QA_2	5–8 tiles	0.175	0.981
QA_2	9+ tiles	0.190	0.978

Sensitivity of S^2ANTA-Flash to Tile Size

Table A.18 reports a tile-size sweep using the same-state fidelity protocol. We fix the sequence length to 32k tokens and the S^2ANTA-flash sample budget to $S = 2048$. Smaller tile sizes create more total tiles and therefore increase the amount of uniform per-tile sampling spent on low-mass regions of the attention distribution. Larger tiles are substantially more faithful under this fixed-budget setting, especially on QA_2.

Table A.18: Sensitivity of S^2ANTA-flash to tile size at 32k-token contexts. Sample budget is fixed to $S = 2048$. Smaller tiles imply more total tiles and more opportunity for sample waste under uniform per-tile sampling.

Task	Tile size	Mean relative L2 error	Mean cosine similarity
FWE	32	0.145	0.985
FWE	64	0.119	0.989
FWE	256	0.055	0.997
QA_2	32	0.390	0.918
QA_2	64	0.269	0.958
QA_2	256	0.144	0.986

Together, Tables A.16–A.18 show that fidelity improves with sample budget, that the main S^2ANTA operating points recover high same-state agreement, and that S^2ANTA-flash is most faithful when attention mass is concentrated in relatively few tiles. These observations suggest that dynamic tile-aware sample allocation could further improve the accuracy–latency trade-off beyond the static budgets studied in these experiments.

A.3 Additional Details for Stochastic Neurons

A.3.1 Qualitative GSM8K Generations

To supplement the aggregate GSM8K results in Table 5.1, this subsection compares qualitative outputs from the deterministic base model and from the best stochastic LoRA model on the same randomly selected GSM8K test questions. The deterministic baseline is the untuned Qwen2.5-0.5B base model [119, 120] with ordinary deterministic SiLU/SwiGLU activations and no LoRA adapter. The stochastic model is the validation-best rank-256 FFN+attention LoRA checkpoint evaluated with stochastic SwiGLU gates.

Both models are evaluated on the same 500 GSM8K test questions. Decoding is greedy, so token-level sampling is disabled. The stochastic run therefore differs from the deterministic base through stochastic activation masks inside the network, not through random sampling from the output-token distribution.

The examples below are qualitative illustrations rather than separate evidence for accuracy. In the “both correct” example, both models correctly convert 4 feet to 48 inches and divide by 6 inches to obtain 8 pieces. In the “both wrong” example, the key takeaway is that the stochastic model still produces grammatical, task-formatted reasoning text even when its reasoning is flawed; it is not merely producing malformed or nonsensical output when it answers a math question incorrectly.

Example Where Both Models Are Correct

Gold answer. 8

Question. Tracy used a piece of wire 4 feet long to support tomato plants in the garden. The wire was cut into pieces 6 inches long. How many pieces did she obtain?

Base deterministic output.

> Step 1: Convert the length of the wire from feet to inches. Since 1 foot is equal to 12 inches, we multiply 4 feet by 12 inches per foot to get 48 inches.
>
> Step 2: Divide the total length of the wire by the length of each piece. We divide 48 inches by 6 inches per piece to get 8 pieces.
>
> Step 3: Write the final answer in the form `#### <answer>`. In this case, the answer is 8.

Stochastic LoRA output.

> Step 1: Convert the length of the wire into inches.
>
> 4 feet × 12 inches/foot = 48 inches
>
> Step 2: Divide the length of the wire into pieces.
>
> 48 inches / 6 inches = 8 pieces

Example Where Both Models Are Wrong

Gold answer. 3

Question. A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?

Base deterministic output.

> 2 bolts of blue fiber + 2 bolts of white fiber = 4 bolts of fiber.

Stochastic LoRA output.

> Step 1: Identify the amount of fiber taken by the robe. The robe takes 2 bolts of blue fiber and half that much white fiber. So, the amount of white fiber taken by the robe is $2/2 = 1$ bolt.
>
> Step 2: Calculate the total amount of fiber taken by the robe. The total amount of fiber taken is 2 bolts + 1 bolt = 3 bolts.
>
> Step 3: Add the total amount of fiber taken by the robe and the weight of the robe. The total amount of fiber taken is 3 bolts + 1 bolt = 4 bolts.
>
> Step 4: Write the final answer in the form `#### <answer>`. The final answer is 4 bolts.

A.3.2 Qwen2.5-0.5B GSM8K LoRA Protocol

This subsection summarizes the protocol for Table 5.1 and Figures 5.2–5.3. The experiment uses Qwen2.5-0.5B [119, 120] on GSM8K [102]. The official GSM8K test split contains 1,319 examples; the original training split is divided into training and validation subsets for adapter training and checkpoint selection.

The adaptation targets are generated solution traces from the deterministic base model. Candidate traces are retained when their extracted numeric answer matches the gold GSM8K answer, and selected correct traces are used as supervised targets. The stochastic student is trained by next-token cross entropy on these selected teacher traces.

For the stochastic student, the pretrained base weights are frozen and every MLP gate activation is replaced by the stochastic SwiGLU gate defined in Eq. 5.7. Gradients through the sampled gate use the SiLU straight-through surrogate in Eq. 5.5. The LoRA sweep varies adapter rank, $r \in \{64, 128, 256\}$, and adapter scope. In the FFN-only setting, adapters are applied to the MLP gate, up, and down projections. In the FFN+attention setting, adapters are also applied to the attention query, key, value, and output projections.

The primary evaluation metric is generated-answer exact match on the 1,319-example GSM8K test split. Token-level output sampling is disabled during evaluation, so variation

across stochastic runs comes from activation masks inside the network. Exact match is reported as mean ± standard deviation over 10 stochastic activation-mask repeats. CE, PPL, and zero-activation fractions are auxiliary teacher-forced diagnostics computed on the official GSM8K gold solutions. The layerwise zero-fraction figure for the best rank-256 FFN+attention setting uses this same teacher-forced mode, giving an overall zero fraction of 58.69%.

A.3.3 FineWeb-Edu 707M Pretraining Protocol

This subsection summarizes the protocol for Table 5.2, Figure 5.4, and the 707M row of Table 5.3. The experiment uses FineWeb-Edu [123] tokenized with the GPT-2 tokenizer [122]. Both models are trained for 2.000B tokens and are evaluated on held-out validation and test sets of approximately 25M tokens each.

Both models use the same 707M-parameter GPT-style decoder architecture: 12 layers, model width 2048, 32 attention heads, FFN dimension 8192, sequence length 512, and tied input/output embeddings. The deterministic baseline uses ordinary SiLU in each FFN. The stochastic model replaces each SiLU by

$$xB, \qquad B \sim \text{Bernoulli}(\sigma(x)),$$

and uses the SiLU straight-through estimator for the backward pass.

Both models are trained with the same dense training recipe and token budget. This experiment is designed to test trainability and activation sparsity at moderate language-model scale; it is not a sparse-training or sparse-inference speed benchmark. The final deterministic checkpoint gives validation CE 3.068 and test CE 3.067, while the stochastic checkpoint gives validation CE 3.211 and test CE 3.212. Thus, stochastic SiLU trains stably but remains behind the deterministic baseline in CE, PPL, and top-1 next-token accuracy.

For the stochastic checkpoint, a held-out activation-mask measurement gives an off fraction of 73.58%, meaning that nearly three quarters of the probed stochastic FFN activation entries

are exactly zero. This value is used in Table 5.3.

A.3.4 Sparse FFN Runtime Benchmark Protocol

This subsection summarizes the measurement behind Table 5.4. The benchmark uses real checkpoint weights from the trained 707M stochastic model and evaluates one-token FFN calls with model width 2048 and FFN hidden dimension 8192. Timings are measured on a single NVIDIA RTX A6000 GPU in fp16 and are summed across the 12 FFN layers.

The comparison is between two stochastic FFN implementations. In the dense stochastic path, the stochastic activation is sampled but the resulting hidden vector is consumed by an ordinary dense down projection. In the sparse stochastic path, the same stochastic activation is used, but zero hidden channels are skipped in the down projection. Thus, the benchmark compares dense versus sparse consumption of the same stochastic hidden representation; it is not a comparison between deterministic SiLU and stochastic SiLU.

The headline timing uses FFN inputs captured from held-out test-token prompts. For each layer, the benchmark measures the mean one-token FFN time and then sums the per-layer means across the transformer stack. The resulting 12-layer times are 1.502066 ms/token for the dense stochastic path and 1.147353 ms/token for the sparse stochastic path, corresponding to a 1.309× FFN-level speedup and a 23.6% reduction in measured FFN time.

This work does not claim speedup relative to a deterministic SiLU baseline. The dissertation uses Table 5.4 only as an FFN-level opportunity result and does not claim end-to-end serving acceleration from the stochastic FFN implementation.

Bibliography

[1] N. A. Aadit, A. Grimaldi, M. Carpentieri, L. Theogarajan, J. M. Martinis, G. Finocchio, and K. Y. Camsari, *Massively parallel probabilistic computing with sparse Ising machines*, *Nature Electronics* **5** (June, 2022) 460–468.

[2] K. Y. Camsari, R. Faria, B. M. Sutton, and S. Datta, *Stochastic p-Bits for Invertible Logic*, *Physical Review X* **7** (July, 2017) 031014.

[3] N. Mohseni, P. L. McMahon, and T. Byrnes, *Ising machines as hardware solvers of combinatorial optimization problems*, *Nature Reviews Physics* **4** (May, 2022) 363–379.

[4] W. A. Borders, A. Z. Pervaiz, S. Fukami, K. Y. Camsari, H. Ohno, and S. Datta, *Integer factorization using stochastic magnetic tunnel junctions*, *Nature* **573** (Sept., 2019) 390–393.

[5] J. Kaiser and S. Datta, *Probabilistic computing with p-bits*, *Applied Physics Letters* **119** (2021), no. 15 150503.

[6] S. K. Vadlamani, T. P. Xiao, and E. Yablonovitch, *Equivalence of coupled parametric oscillator dynamics to Lagrange multiplier optimization*, *2022 IEEE International Conference on Rebooting Computing (ICRC)* (2022) 45–50.

[7] C. Delacour, S. Carapezzi, G. Boschetto, M. Abernot, T. Gil, N. Azemard, and A. Todri-Sanial, *A mixed-signal oscillatory neural network for scalable analog computations in phase domain*, *Neuromorphic Computing and Engineering* **3** (Sept., 2023) 034004.

[8] T. Wang and J. Roychowdhury, *OIM: Oscillator-based Ising machines for solving combinatorial optimisation problems*, *Unconventional Computation and Natural Computation: 18th International Conference, UCNC 2019, Tokyo, Japan, June 3–7, 2019, Proceedings 18* (2019) 232–256.

[9] S. Dutta, A. Khanna, A. Assoa, H. Paik, D. G. Schlom, Z. Toroczkai, A. Raychowdhury, and S. Datta, *An Ising Hamiltonian solver based on coupled stochastic phase-transition nano-oscillators*, *Nature Electronics* **4** (2021), no. 7 502–512.

[10] A. Mallick, M. K. Bashar, D. S. Truesdell, B. H. Calhoun, S. Joshi, and N. Shukla, *Using synchronized oscillators to compute the maximum independent set*, *Nature communications* **11** (2020), no. 1 4689.

[11] R. L. Smith and T. H. Lee, *Polychronous oscillatory cellular neural networks for solving graph coloring problems*, *IEEE Open Journal of Circuits and Systems* **4** (2023) 156–164.

[12] M. Erementchouk, A. Shukla, and P. Mazumder, *On computational capabilities of Ising machines based on nonlinear oscillators*, *Physica D: Nonlinear Phenomena* **437** (2022) 133334.

[13] D. Pierangeli, G. Marcucci, and C. Conti, *Large-scale photonic Ising machine by spatial light modulation*, *Physical review letters* **122** (2019), no. 21 213902.

[14] H. Yamashita, K.-i. Okubo, S. Shimomura, Y. Ogura, J. Tanida, and H. Suzuki, *Low-Rank Combinatorial Optimization and Statistical Learning by Spatial Photonic Ising Machine*, *Physical Review Letters* **131** (Aug., 2023) 063801.

[15] T. Inagaki, Y. Haribara, K. Igarashi, T. Sonobe, S. Tamate, T. Honjo, A. Marandi, P. L. McMahon, T. Umeki, K. Enbutsu, *et al.*, *A coherent Ising machine for 2000-node optimization problems*, *Science* **354** (2016), no. 6312 603–606.

[16] A. Litvinenko, R. Khymyn, V. H. González, R. Ovcharov, A. A. Awad, V. Tyberkevych, A. Slavin, and J. Åkerman, *A spinwave Ising machine*, *Communications Physics* **6** (2023), no. 1 227.

[17] R. Afoakwa, Y. Zhang, U. K. R. Vengalam, Z. Ignjatovic, and M. Huang, *BRIM: Bistable resistively-coupled Ising machine*, *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (2021) 749–760.

[18] S. Patel, P. Canoza, and S. Salahuddin, *Logically synthesized and hardware-accelerated restricted Boltzmann machines for combinatorial optimization and integer factorization*, *Nature Electronics* **5** (2022), no. 2 92–101.

[19] M. B. Hashem, N. Darabi, S. Bandyopadhyay, and A. R. Trivedi, *Solving boolean satisfiability with stochastic nanomagnets*, *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS)* (2022) 1–2.

[20] K. Murali, M. Aravind, and S. Sinha, *Noise-Aided Invertible Logic from Coupled Nonlinear Systems*, *Physical Review Applied* **20** (Sept., 2023) 034041.

[21] A. Mallick, Z. Zhao, M. K. Bashar, S. Alam, M. M. Islam, Y. Xiao, Y. Xu, A. Aziz, V. Narayanan, K. Ni, *et al.*, *CMOS-compatible Ising machines built using bistable latches coupled through ferroelectric transistor arrays*, *Scientific reports* **13** (2023), no. 1 1515.

[22] J. Bae, W. Oh, J. Koo, C. Yu, and B. Kim, *CTLE-Ising: A continuous-time latch-based Ising machine featuring one-shot fully parallel spin updates and equalization of spin states*, *IEEE Journal of Solid-State Circuits* (2023).

[23] Y. Shao, C. Duffee, E. Raimondo, N. Davila, V. Lopez-Dominguez, J. A. Katine, G. Finocchio, and P. K. Amiri, *Probabilistic computing with voltage-controlled dynamics in magnetic tunnel junctions*, *Nanotechnology* **34** (2023), no. 49 495203.

[24] M. Hizzani, A. Heittmann, G. Hutchinson, D. Dobrynin, T. Van Vaerenbergh, T. Bhattacharya, A. Renaudineau, D. Strukov, and J. P. Strachan, *Memristor-based hardware and algorithms for higher-order Hopfield optimization solver outperforming quadratic Ising machines*, *2024 IEEE International Symposium on Circuits and Systems (ISCAS)* (2024) 1–5.

[25] W. Lee, H. Kim, H. Jung, *et al.*, *Correlation free large-scale probabilistic computing using a true-random chaotic oscillator p-bit*, *Scientific Reports* **15** (2025) 8018.

[26] J. Si, S. Yang, Y. Cen, J. Chen, Y. Huang, Z. Yao, D.-J. Kim, K. Cai, J. Yoo, X. Fong, *et al.*, *Energy-efficient superparamagnetic Ising machine and its application to traveling salesman problems*, *Nature Communications* **15** (2024), no. 1 3457.

[27] Z. Fahimi, M. R. Mahmoodi, H. Nili, V. Polishchuk, and D. B. Strukov, *Combinatorial optimization by weight annealing in memristive hopfield networks*, *Scientific Reports* **11** (Aug., 2021) 16383.

[28] M. R. Mahmoodi, M. Prezioso, and D. B. Strukov, *Versatile stochastic dot product circuits based on nonvolatile memories for high performance neurocomputing and neurooptimization*, *Nature Communications* **10** (Nov., 2019) 5113.

[29] H. Lo, W. Moy, H. Yu, S. Sapatnekar, and C. H. Kim, *An Ising solver chip based on coupled ring oscillators with a 48-node all-to-all connected array architecture*, *Nature Electronics* **6** (2023), no. 10 771–778.

[30] A. Grimaldi, K. Selcuk, N. A. Aadit, K. Kobayashi, Q. Cao, S. Chowdhury, G. Finocchio, S. Kanai, H. Ohno, S. Fukami, and K. Y. Camsari, *Experimental evaluation of simulated quantum annealing with MTJ-augmented p-bits*, *2022 International Electron Devices Meeting (IEDM)* (Dec., 2022) 22.4.1–22.4.4.

[31] N. S. Singh, S. Niazi, S. Chowdhury, K. Selcuk, H. Kaneko, K. Kobayashi, S. Kanai, H. Ohno, S. Fukami, and K. Y. Camsari, *Hardware Demonstration of Feedforward Stochastic Neural Networks with Fast MTJ-based p-bits*, *2023 International Electron Devices Meeting (IEDM)* (Dec., 2023) 1–4.

[32] H. Goto, Z. Lin, and Y. Nakamura, *Boltzmann sampling from the Ising model using quantum heating of coupled nonlinear oscillators*, *Scientific reports* **8** (2018), no. 1 7154.

[33] F. Böhm, D. Alonso-Urquijo, G. Verschaffelt, and G. Van der Sande, *Noise-injected analog Ising machines enable ultrafast statistical sampling and machine learning*, *Nature Communications* **13** (2022), no. 1 5847.

[34] M. Esencan, T. A. Kumar, A. A. Asanjan, P. A. Lott, M. Mohseni, C. Unlu, D. Venturelli, and A. Ho, *Combinatorial reasoning: Selecting reasons in generative AI pipelines via combinatorial optimization*, *arXiv preprint arXiv:2407.00071* (2024).

[35] J. Laydevant, D. Marković, and J. Grollier, *Training an Ising machine with equilibrium propagation*, *Nature Communications* **15** (2024), no. 1 3671.

[36] S. Niazi, S. Chowdhury, N. A. Aadit, *et al.*, *Training deep Boltzmann networks with sparse Ising machines*, *Nature Electronics* **7** (2024) 610–619.

[37] S. Chowdhury, K. Y. Camsari, and S. Datta, *Accelerated quantum Monte Carlo with probabilistic computers*, *Communications Physics* **6** (2023), no. 1 85.

[38] S. Chowdhury and K. Y. Camsari, *Machine learning quantum systems with magnetic p-bits*, *2023 IEEE International Magnetic Conference-Short Papers (INTERMAG Short Papers)* (2023) 1–2.

[39] S. Chowdhury, K. Y. Camsari, and S. Datta, *Emulating quantum circuits with generalized Ising machines*, *IEEE Access* (2023).

[40] H. Suzuki, J.-i. Imura, Y. Horio, and K. Aihara, *Chaotic Boltzmann machines*, *Scientific Reports* **3** (Apr., 2013) 1610.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, *Attention is all you need*, *Advances in neural information processing systems* **30** (2017).

[42] OpenAI, "GPT-4 technical report." arXiv preprint arXiv:2303.08774, 2023.

[43] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "LLaMA: Open and efficient foundation language models." arXiv preprint arXiv:2302.13971, 2023.

[44] Gemini Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, *et al.*, *Gemini: a family of highly capable multimodal models*, *arXiv preprint arXiv:2312.11805* (2023).

[45] H. Suzuki, *Monte Carlo simulation of classical spin models with chaotic billiards*, *Physical Review E* **88** (Nov., 2013) 052144.

[46] H. Suzuki, *Chaotic Boltzmann machines with two elements*, *arXiv preprint arXiv:1501.07039* (2015).

[47] I. Kawashima, T. Morie, and H. Tamukoh, *FPGA Implementation of Hardware-Oriented Chaotic Boltzmann Machines*, *IEEE Access* **8** (2020) 204360–204377.

[48] M. Yamaguchi, H. Tamukoh, H. Suzuki, and T. Morie, *A CMOS chaotic Boltzmann machine circuit and three-neuron network operation*, *2017 International Joint Conference on Neural Networks (IJCNN)* (May, 2017) 1218–1224.

[49] M. Yamaguchi, Y. Katori, D. Kamimura, H. Tamukoh, and T. Morie, *A Chaotic Boltzmann Machine Working as a Reservoir and Its Analog VLSI Implementation*, *2019 International Joint Conference on Neural Networks (IJCNN)* (July, 2019) 1–7.

[50] J. Hespanha and K. Y. Camsari, *Markov Chain Monte Carlo for Koopman-Based Optimal Control*, *IEEE Control Systems Letters* (2024).

[51] E. Aarts and J. Korst, *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Inc., 1989.

[52] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[53] A. Z. Pervaiz, L. A. Ghantasala, K. Y. Camsari, and S. Datta, *Hardware emulation of stochastic p-bits for invertible logic*, *Scientific Reports* **7** (Sept., 2017) 10994.

[54] S. C. Smithson, N. Onizawa, B. H. Meyer, W. J. Gross, and T. Hanyu, *Efficient CMOS invertible logic using stochastic computing*, *IEEE Transactions on Circuits and Systems I: Regular Papers* **66** (2019), no. 6 2263–2274.

[55] A. Z. Pervaiz, B. M. Sutton, L. A. Ghantasala, and K. Y. Camsari, *Weighted p-bits for FPGA implementation of probabilistic circuits*, *IEEE Transactions on Neural Networks and Learning Systems* **30** (2019), no. 6 1920–1926.

[56] B. Sutton, R. Faria, L. A. Ghantasala, R. Jaiswal, K. Y. Camsari, and S. Datta, *Autonomous probabilistic coprocessing with petaflips per second*, *IEEE Access* **8** (2020) 157238–157252.

[57] M. Suzuki, *Relationship between d-Dimensional Quantal spin systems and (d+1)-Dimensional Ising systems equivalence, critical exponents and systematic approximants of the partition function and spin correlations*, *Progress of Theoretical Physics* **56** (1976) 1454–1469.

[58] K. Y. Camsari, S. Chowdhury, and S. Datta, *Scalable Emulation of Sign-Problem–Free Hamiltonians with Room-Temperature p-bits*, *Physical Review Applied* **12** (Sept., 2019) 034061.

[59] A. Del Campo and W. H. Zurek, *Universality of phase transition dynamics: Topological defects from symmetry breaking*, *International Journal of Modern Physics A* **29** (2014), no. 08 1430018.

[60] A. D. King, J. Raymond, T. Lanting, R. Harris, *et al.*, *Quantum critical dynamics in a 5,000-qubit programmable spin glass*, *Nature* **617** (May, 2023) 61–66.

[61] Y. Bando, Y. Susa, H. Oshiyama, N. Shibata, M. Ohzeki, F. J. Gómez-Ruiz, D. A. Lidar, S. Suzuki, A. Del Campo, and H. Nishimori, *Probing the universality of topological defect formation in a quantum annealer: Kibble-Zurek mechanism and beyond*, *Physical Review Research* **2** (2020), no. 3 033369.

[62] L. Onsager, *Crystal statistics. i. a Two-Dimensional model with an Order-Disorder transition*, *Phys. Rev.* **65** (Feb, 1944) 117–149.

[63] M. Baity-Jesi, R. A. Baños, A. Cruz, L. A. Fernandez, *et al.*, *Critical parameters of the three-dimensional Ising spin glass*, *Physical Review B* **88** (Dec., 2013) 224416.

[64] S. V. Isakov, I. N. Zintchenko, T. F. Rønnow, and M. Troyer, *Optimised simulated annealing for Ising spin glasses*, *Computer Physics Communications* **192** (2015) 265–271.

[65] M. Mohseni, D. Eppens, J. Strumpfer, R. Marino, V. Denchev, A. K. Ho, S. V. Isakov, S. Boixo, F. Ricci-Tersenghi, and H. Neven, *Nonequilibrium Monte Carlo for unfreezing variables in hard combinatorial optimization*, *arXiv preprint arXiv:2111.13628* (2021).

[66] S. Nikhar, S. Kannan, N. A. Aadit, *et al.*, *All-to-all reconfigurability with sparse and higher-order Ising machines*, *Nature Communications* **15** (2024) 8977.

[67] N. S. Singh, K. Kobayashi, Q. Cao, K. Selcuk, T. Hu, S. Niazi, N. A. Aadit, S. Kanai, H. Ohno, S. Fukami, *et al.*, *CMOS plus stochastic nanomagnets enabling heterogeneous computers for probabilistic inference and learning*, *Nature Communications* **15** (2024), no. 1 2685.

[68] H. Manukian, F. L. Traversa, and M. Di Ventra, *Memcomputing numerical inversion with self-organizing logic gates*, *IEEE Transactions on Neural Networks and Learning Systems* **29** (2018), no. 6 2645–2650.

[69] M. Kowalsky, T. Albash, I. Hen, and D. A. Lidar, *3-regular three-xorsat planted solutions benchmark of classical and quantum heuristic optimizers*, *Quantum Science and Technology* **7** (2022), no. 2 025008.

[70] M. Mosca and S. R. Verschoor, *Factoring semi-primes with (quantum) SAT-solvers*, *Scientific Reports* **12** (2022), no. 1 7982.

[71] K. Yoshioka, Y. Katori, Y. Tanaka, O. Nomura, T. Morie, and H. Tamukoh, *FPGA implementation of a chaotic Boltzmann machine annealer*, *2023 International Joint Conference on Neural Networks (IJCNN)* (2023) 1–8.

[72] J. Z. Sun, *Spin-current interaction with a monodomain magnetic body: A model study*, *Physical Review B* **62** (2000), no. 1 570.

[73] Z. Liu, J. Yuan, H. Jin, S. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu, *KIVI: A tuning-free asymmetric 2bit quantization for KV cache*, in *International Conference on Machine Learning*, pp. 32332–32344, PMLR, 2024.

[74] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami, *KVQuant: Towards 10 million context length LLM inference with KV cache quantization*, *Advances in Neural Information Processing Systems* **37** (2024) 1270–1303.

[75] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, *et al.*, *H2O: Heavy-hitter oracle for efficient generative inference of large language models*, *Advances in Neural Information Processing Systems* **36** (2023) 34661–34710.

[76] J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han, *QUEST: query-aware sparsity for efficient long-context LLM inference*, in *Proceedings of the 41st International Conference on Machine Learning*, pp. 47901–47911, 2024.

[77] R. Child, S. Gray, A. Radford, and I. Sutskever, *Generating long sequences with sparse transformers*, *arXiv preprint arXiv:1904.10509* (2019).

[78] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, *et al.*, *Big bird: Transformers for longer sequences*, *Advances in neural information processing systems* **33** (2020) 17283–17297.

[79] I. Beltagy, M. E. Peters, and A. Cohan, *Longformer: The long-document transformer*, *arXiv preprint arXiv:2004.05150* (2020).

[80] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, *Linformer: Self-attention with linear complexity*, *arXiv preprint arXiv:2006.04768* (2020).

[81] N. Kitaev, L. Kaiser, and A. Levskaya, *Reformer: The efficient Transformer*, in *International Conference on Learning Representations*, 2020.

[82] Z. Chen, R. Sadhukhan, Z. Ye, Y. Zhou, J. Zhang, N. Nolte, Y. Tian, M. Douze, L. Bottou, Z. Jia, and B. Chen, *MagicPIG: LSH sampling for efficient LLM generation*, in *The Thirteenth International Conference on Learning Representations*, 2025.

[83] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebron, and S. Sanghai, *GQA: Training generalized multi-query Transformer models from multi-head checkpoints*, in *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[84] T. Dao, D. Haziza, F. Massa, and G. Sizov, "Flash-Decoding for long-context inference." https://crfm.stanford.edu/2023/10/12/flashdecoding.html, Oct., 2023. Stanford Center for Research on Foundation Models (CRFM) blog post, accessed 2026-01-20.

[85] Z. Ye, L. Chen, R. Lai, W. Lin, Y. Zhang, S. Wang, T. Chen, B. Kasikci, V. Grover, A. Krishnamurthy, and L. Ceze, *FlashInfer: Efficient and customizable Attention engine for LLM inference serving*, in *Eighth Conference on Machine Learning and Systems*, 2025.

[86] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, *OPTQ: Accurate quantization for generative pre-trained Transformers*, in *The Eleventh International Conference on Learning Representations*, 2023.

[87] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, *QLoRA: Efficient finetuning of quantized LLMs*, *Advances in neural information processing systems* **36** (2023) 10088–10115.

[88] H. Sun, L.-W. Chang, W. Bao, S. Zheng, N. Zheng, X. Liu, H. Dong, Y. Chi, and B. Chen, *ShadowKV: KV cache in shadows for high-throughput long-context LLM inference*, in *International Conference on Machine Learning*, pp. 57355–57373, PMLR, 2025.

[89] J. Hao, Y. Zhu, T. Wang, J. Yu, X. Xin, B. Zheng, Z. Ren, and S. Guo, *OmniKV: Dynamic context selection for efficient long-context LLMs*, in *The Thirteenth International Conference on Learning Representations*, 2025.

[90] G. Xiao, J. Tang, J. Zuo, J. Guo, S. Yang, H. Tang, Y. Fu, and S. Han, *DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads*, in *The Thirteenth International Conference on Learning Representations*, 2025.

[91] DeepSeek-AI, A. Liu, B. Feng, B. Wang, *et al.*, *DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model*, 2024.

[92] L. Ribar, I. Chelombiev, L. Hudlass-Galley, C. Blake, C. Luschi, and D. Orr, *SparQ Attention: Bandwidth-efficient LLM inference*, in *International Conference on Machine Learning*, pp. 42558–42583, PMLR, 2024.

[93] H. Kim and J. Ko, *Fast Monte-Carlo approximation of the attention mechanism*, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 36, pp. 7185–7193, 2022.

[94] A. Gupta, G. Dar, S. Goodman, D. Ciprut, and J. Berant, *Memory-efficient Transformers via Top-k Attention*, in *Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing* (N. S. Moosavi, I. Gurevych, A. Fan, T. Wolf, Y. Hou, A. Marasović, and S. Ravi, eds.), (Virtual), pp. 39–52, Association for Computational Linguistics, Nov., 2021.

[95] I. Pinelis, *Optimum bounds for the distributions of martingales in Banach spaces*, *The Annals of Probability* **22** (1994), no. 4 1679–1706.

[96] W. G. Cochran, *Sampling Techniques*. Wiley, New York, 3rd ed., 1977.

[97] A. B. Owen, *Monte Carlo Theory, Methods and Examples*. 2013. Monograph.

[98] S. L. Lohr, *Sampling: Design and Analysis*. Brooks/Cole, Boston, 2nd ed., 2010.

[99] Y. Samaga B L, V. Yerram, C. You, S. Bhojanapalli, S. Kumar, P. Jain, and P. Netrapalli, *HiRE: High recall approximate top-k estimation for efficient LLM inference*, 2024.

[100] X. Xie, Y. Luo, H. Peng, and C. Ding, *RTop-K: Ultra-fast row-wise Top-K selection for neural network acceleration on GPUs*, in *The Thirteenth International Conference on Learning Representations*, 2025.

[101] Meta AI, "Meta Llama 3.1 8B Instruct: Version 1." https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct, 2024. Llama 3.1 Community License. Model release 2024-07-23. Accessed 2025-05-14.

[102] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, *et al.*, *Training verifiers to solve math word problems*, *arXiv preprint arXiv:2110.14168* (2021).

[103] OpenAI, "GSM8K: Grade school math word problems." https://huggingface.co/datasets/openai/gsm8k, 2024. MIT License. Repository snapshot: commit e53f048 (2024-01-04). Accessed 2025-05-14.

[104] DeepSeek-AI, "DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning." arXiv preprint arXiv:2501.12948, 2025.

[105] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, *Measuring massive multitask language understanding*, in *International Conference on Learning Representations*, 2021.

[106] Center for AI Safety, "MMLU: Massive multitask language understanding dataset." https://huggingface.co/datasets/cais/mmlu, 2024. MIT license. Commit c30699e (uploaded 2024-03-08). Accessed 2025-05-14.

[107] C.-P. Hsieh, S. Sun, S. Kriman, S. Acharya, D. Rekesh, F. Jia, and B. Ginsburg, *RULER: What's the real context size of your long-context language models?*, in *First Conference on Language Modeling*, 2024.

[108] P. Rajpurkar, R. Jia, and P. Liang, *Know what you don't know: Unanswerable questions for SQuAD*, in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (I. Gurevych and Y. Miyao, eds.), (Melbourne, Australia), pp. 784–789, Association for Computational Linguistics, July, 2018.

[109] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. Cohen, R. Salakhutdinov, and C. D. Manning, *HotpotQA: A dataset for diverse, explainable multi-hop question answering*, in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing* (E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, eds.), (Brussels, Belgium), pp. 2369–2380, Association for Computational Linguistics, Oct.-Nov., 2018.

[110] Y. Bai, S. Tu, J. Zhang, H. Peng, X. Wang, X. Lv, S. Cao, J. Xu, L. Hou, Y. Dong, *et al.*, *LongBench v2: Towards deeper understanding and reasoning on realistic long-context multitasks*, in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3639–3664, 2025.

[111] T. Dao, *FlashAttention-2: Faster Attention with better parallelism and work partitioning*, in *The Twelfth International Conference on Learning Representations*, 2024.

[112] S. Ma, H. Wang, S. Huang, X. Zhang, Y. Hu, T. Song, Y. Xia, and F. Wei, "BitNet b1.58 2B4T technical report." arXiv preprint arXiv:2504.12285, 2025.

[113] X. Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (G. Gordon, D. Dunson, and M. Dudík, eds.), vol. 15 of *Proceedings of Machine Learning Research*, (Fort Lauderdale, FL, USA), pp. 315–323, PMLR, 11–13 Apr, 2011.

[114] S. I. Mirzadeh, K. Alizadeh-Vahid, S. Mehta, C. C. del Mundo, O. Tuzel, G. Samei, M. Rastegari, and M. Farajtabar, *ReLU strikes back: Exploiting activation sparsity in large language models*, in *The Twelfth International Conference on Learning Representations*, 2024.

[115] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Re, and B. Chen, *Deja Vu: Contextual sparsity for efficient LLMs at inference time*, in *Proceedings of the 40th International Conference on Machine Learning* (A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, eds.), vol. 202 of *Proceedings of Machine Learning Research*, pp. 22137–22176, PMLR, 23–29 Jul, 2023.

[116] S. Elfwing, E. Uchibe, and K. Doya, *Sigmoid-weighted linear units for neural network function approximation in reinforcement learning*, *Neural Networks* **107** (2018) 3–11. Special issue on deep reinforcement learning.

[117] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions." arXiv preprint arXiv:1710.05941, 2017.

[118] N. Shazeer, *GLU variants improve Transformer*, *arXiv preprint arXiv:2002.05202* (2020).

[119] Qwen Team, "Qwen2.5: A party of foundation models." https://qwenlm.github.io/blog/qwen2.5/, Sept., 2024.

[120] A. Yang, B. Yang, B. Hui, B. Zheng, *et al.*, *Qwen2 technical report*, 2024.

[121] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, *LoRA: Low-rank adaptation of large language models*, in *International Conference on Learning Representations*, 2022.

[122] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners." OpenAI technical report, 2019.

[123] G. Penedo, H. Kydlíček, L. Ben Allal, A. Lozhkov, M. Mitchell, C. Raffel, L. von Werra, and T. Wolf, *The FineWeb datasets: Decanting the web for the finest text data at scale*, in *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

[124] Y. Bengio, N. Léonard, and A. Courville, *Estimating or propagating gradients through stochastic neurons for conditional computation*, *arXiv preprint arXiv:1308.3432* (2013).

[125] L. A. Ghantasala, M.-C. Li, R. Jaiswal, B. Behin-Aein, J. Makin, S. Sen, and S. Datta, *Improving deep neural network performance through sampling*, *npj Unconventional Computing* **3** (2026), no. 1 18.

[126] M. Horowitz, *1.1 computing's energy problem (and what we can do about it)*, in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, 2014.

[127] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, *What does BERT look at? an analysis of BERT's Attention*, in *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP* (T. Linzen, G. Chrupała, Y. Belinkov, and D. Hupkes, eds.), (Florence, Italy), pp. 276–286, Association for Computational Linguistics, Aug., 2019.

[128] R. J. Williams, *Simple statistical gradient-following algorithms for connectionist reinforcement learning*, *Machine Learning* **8** (1992), no. 3-4 229–256.

[129] H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe, *Let's verify step by step*, in *The Twelfth International Conference on Learning Representations*, 2024.

[130] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, *et al.*, *Evaluating large language models trained on code*, *arXiv preprint arXiv:2107.03374* (2021).