Stabilizing Quantized KV Caches via Subspace Embeddings and Manifold Interpolation

Indrajeet Aditya Roy [1]

Abstract

Large Language Models (LLMs) function as autoregressive predictors, computing attention over a cached history of key–value projections to avoid the $O(T^2)$ cost of recomputation (32; 19). As context lengths grow, memory constraints necessitate aggressive quantization of these caches to low-precision integer representations (25; 16).

This quantization introduces a *metric mismatch* between storage and semantic spaces: perturbations measured by Hamming distance in the storage domain do not correspond to bounded perturbations in Euclidean distance after dequantization (10). Consequently, bit-level errors manifest as sparse, large-magnitude impulse vectors that can collapse the attention distribution onto corrupted entries, causing catastrophic model failure.

A linear-algebraic framework is proposed for stabilizing quantized KV caches using classical error-correcting codes (26). Quantized values are embedded into higher-dimensional subspaces over $\mathbb{F}_2$ via the generator matrices of Hamming and Golay codes, enabling syndrome-based detection and correction of bit-flip errors before the values are lifted back to $\mathbb{R}^d$ (17). For errors beyond the correction capacity, a geometric recovery strategy is employed, exploiting the local smoothness of the cache sequence through linear interpolation of detected erasures.

Experiments on GPT-2 and LLaMA-3.1-8B (28; 11) demonstrate that at bit error rates up to 10^{-2}, unprotected caches diverge catastrophically, while protected caches maintain baseline performance.

[1]Mathematics Department, College of Sciences, Northeastern University, Boston, MA, USA. Correspondence to: Indrajeet Aditya Roy <roy.i@northeastern.edu>.

1. Introduction

Large Language Models (LLMs) can be viewed as high-dimensional dynamical systems governed by linear algebraic operations. These systems function as *autoregressive* predictors: they generate sequences recursively, where the output of step t conditions the computation at step $t + 1$ (28).

From a mathematical standpoint, this recursion creates a distinct complexity challenge. At every time step, the model must evaluate inner products between a current query vector and the entire history of key vectors. Naively recomputing these projections at every step implies a complexity of $\mathcal{O}(T^2)$ for a sequence of length T.

To resolve this, implementations employ a caching strategy known as the *Key–Value (KV) cache* (19). Rather than recomputing projections over the entire history, the system stores the projected vectors in two growing matrices, $K, V \in \mathbb{R}^{T\times d}$. Each new token requires only appending one row and computing a single matrix–vector product, reducing per-step complexity to $\mathcal{O}(T)$. However, as context lengths grow ($T \to 10^5$), the memory required to store these matrices in high-precision floating point becomes intractable (25).

The standard solution is quantization: a discretization map that projects continuous vectors in $\mathbb{R}^d$ onto low-precision integer representations (e.g., 4-bit integers with 16 discrete levels) (12). While this compression reduces memory by $4\times$ relative to FP16, it fundamentally alters the algebraic structure of the cached representations.

In high-precision arithmetic, quantization error is typically modeled as small-magnitude additive noise ($\epsilon \sim \mathcal{N}(0, \sigma^2)$), which averages out in high-dimensional inner products. In the regime of 4-bit integers, however, a different failure mode emerges. Due to the weighted positional encoding of binary representations, a single bit-flip does not act as Gaussian noise—it acts as a sparse, large-magnitude impulse in the vector space (10). A flip in the most significant bit of an INT4 value can shift it by ± 8 quantization levels, inducing a perturbation comparable to the full dynamic range. When such corrupted vectors participate in attention computation, the exponential amplification of softmax con-

centrates probability mass on corrupted entries, causing the output distribution to collapse (35).

Such bit-level perturbations arise from multiple sources in practice: DRAM soft errors from cosmic radiation, voltage noise in aggressive low-power regimes, and approximation errors in emerging compute-in-memory architectures (17). As quantization pushes representations toward their information-theoretic limits, the margin for such perturbations shrinks, making error resilience increasingly critical.

This work addresses this instability as a problem of *linear algebra over finite fields*. The core insight is that while attention computation operates over $\mathbb{R}^d$, the integrity of quantized cache entries is naturally modeled over the binary vector space $\mathbb{F}_2^n$. This separation motivates the application of classical error-correcting codes (ECC) as an algebraic protection layer (26).

A framework is proposed in which quantized values are embedded into higher-dimensional subspaces defined by the generator matrices of Hamming and Golay codes. The null-space characterization provided by parity-check matrices enables $O(1)$ syndrome-based detection and correction of bit errors before values are lifted back to $\mathbb{R}^d$. For errors exceeding the code's correction capacity, a geometric recovery strategy is employed that exploits the local smoothness of cache sequences through linear interpolation of detected erasures—bridging discrete coding theory with continuous manifold assumptions.

Contributions. This work makes the following contributions:

1. The *metric mismatch* problem in quantized KV caches is formalized, demonstrating how bit-level errors in Hamming space induce unbounded perturbations in Euclidean space, leading to catastrophic attention collapse.
2. GPU-accelerated Triton kernels are developed for Hamming(7,4), Hamming(8,4) SECDED, and Golay(24,12) encoding and decoding, fused directly into the attention computation to minimize overhead.
3. *Manifold-aware interpolation* is introduced for recovering SECDED-detected double errors, demonstrating that this lightweight geometric heuristic achieves robustness equivalent to Golay's algebraically stronger three-error correction.
4. The framework is validated on GPT-2 and LLaMA-3.1-8B, showing that at bit error rates up to 10^{-2}, unprotected INT4 caches fail catastrophically (perplexity > 1000), while protected caches maintain baseline performance with zero catastrophic failures across all trials.

2. Background

2.1. The Linear Algebra of Attention

Large Language Models (LLMs) are composed of stacked *Transformer layers*, each applying a sequence of linear algebraic operations followed by non-linear activations. The computational core of each layer is the *Attention Mechanism* (32), which acts as a data-dependent aggregation operator, computing a new state vector as a convex combination of historical vectors.

Let the input at position t be represented by a vector $\mathbf{x} \in \mathbb{R}^d$. The layer applies three parallel linear transformations parameterized by weight matrices $W_Q, W_K, W_V \in \mathbb{R}^{d \times d_k}$, mapping the input into query, key, and value representations:

$$\mathbf{q} = \mathbf{x}W_Q, \quad \mathbf{k} = \mathbf{x}W_K, \quad \mathbf{v} = \mathbf{x}W_V, \tag{1}$$

where d_k is the per-head dimension (typically $d_k = d/h$ for h attention heads). For clarity, the single-head case is presented; multi-head attention applies this mechanism in parallel across h independent subspaces.

2.1.1. Attention Score Computation

The system maintains a history of the sequence through position t. Let $K \in \mathbb{R}^{t \times d_k}$ and $V \in \mathbb{R}^{t \times d_k}$ be matrices whose i-th rows contain the key and value vectors from position i.

The attention mechanism computes a score vector $\mathbf{s} \in \mathbb{R}^t$ by evaluating scaled inner products between the current query and all historical keys:

$$\mathbf{s} = \frac{\mathbf{q}K^\top}{\sqrt{d_k}}, \tag{2}$$

where the $\sqrt{d_k}$ scaling prevents dot products from growing with dimension, stabilizing gradient flow during training (32). Each component $s_i = \langle \mathbf{q}, \mathbf{k}_i \rangle / \sqrt{d_k}$ measures the alignment between the current query and the i-th historical key.

In autoregressive generation, a causal mask ensures that position t attends only to positions $1, \ldots, t$, enforcing the left-to-right dependency structure.

2.1.2. Softmax Normalization

The score vector $\mathbf{s}$ is mapped to the probability simplex $\Delta^{t-1} = \boldsymbol{\alpha} \in \mathbb{R}^t : \alpha_i \geq 0, \sum_i \alpha_i = 1$ via the softmax function:

$$\boldsymbol{\alpha} = \mathrm{softmax}(\mathbf{s}), \quad \text{where } \alpha_i = \frac{\exp(s_i)}{\sum_{j=1}^{t} \exp(s_j)}. \tag{3}$$

This normalization converts unbounded scores into a valid probability distribution over historical positions.

2.1.3. Output as Convex Combination

The output vector $\mathbf{z} \in \mathbb{R}^{d_k}$ is computed as the $\boldsymbol{\alpha}$-weighted sum of value vectors:

$$\mathbf{z} = \boldsymbol{\alpha}^\top V = \sum_{i=1}^{t} \alpha_i \mathbf{v}_i. \tag{4}$$

Since $\boldsymbol{\alpha} \in \Delta^{t-1}$, the output lies in the convex hull of $\mathbf{v}_1, \ldots, \mathbf{v}_t$. When query $\mathbf{q}$ is orthogonal to key $\mathbf{k}_i$, the corresponding weight $\alpha_i \to 0$ and value $\mathbf{v}_i$ contributes negligibly. Conversely, high query–key alignment concentrates weight on the corresponding value.

Vulnerability. The vulnerability of interest arises from the storage of K and V. In memory-constrained deployments, these matrices are quantized to low-precision formats (e.g., INT4) rather than stored in FP16. A bit-level perturbation to an entry in K alters the inner product $\mathbf{q}K^\top$, potentially causing the softmax to concentrate probability mass on a corrupted entry. The exponential nature of softmax amplifies even moderate score perturbations: if a corrupted key produces an anomalously high score, nearly all attention weight shifts to the corresponding (potentially irrelevant) value, destabilizing the output (35).

2.2. Recursive Matrix Formulation (The KV Cache)

In a direct implementation of autoregressive attention, the score vector $\mathbf{s}$ would require computing inner products against the entire history at every time step. At step t, this costs $\mathcal{O}(t \cdot d)$ operations. Summing over all steps yields a cumulative complexity of $\sum_{t=1}^{T} \mathcal{O}(t \cdot d) = \mathcal{O}(T^2 d)$, which becomes intractable for long sequences.

To resolve this, the computation is reformulated recursively. Since past key-value pairs are immutable during autoregressive generation, the system caches them rather than recomputing. At step t, only the new vectors $\mathbf{k}_t$ and $\mathbf{v}_t$ are computed and appended:

$$K_t = \begin{bmatrix} K_{t-1} \\ \mathbf{k}_t \end{bmatrix}, \qquad V_t = \begin{bmatrix} V_{t-1} \\ \mathbf{v}_t \end{bmatrix}. \tag{5}$$

This *Key–Value (KV) cache* (19) reduces per-step complexity to $\mathcal{O}(T \cdot d)$: a single matrix-vector product against the cached history.

Memory Bottleneck. The KV cache transforms the computational bottleneck into a memory bottleneck. For a model with L layers, h KV heads, and head dimension d_k, the cache stores $2 \cdot L \cdot h \cdot T \cdot d_k$ values. Table 1 illustrates this tradeoff. For a standard 8B parameter model using Grouped Query Attention (GQA), a modest batch size of 4 fills an entire A100 GPU at 128k context.

Context Length (T)	Cumulative FLOPs (no cache)	KV Cache (FP16)	KV Cache (INT4)
1,000	$\mathcal{O}(10^6)$	0.5 GB	0.13 GB
10,000	$\mathcal{O}(10^8)$	5.0 GB	1.25 GB
100,000	$\mathcal{O}(10^{10})$	50.0 GB	12.5 GB
128,000	$\mathcal{O}(10^{10})$	64.0 GB	16.0 GB

Table 1. **Computational vs. Storage Complexity.** Without caching, autoregressive generation has $\mathcal{O}(T^2)$ cumulative complexity. The KV cache reduces this to $\mathcal{O}(T)$ per step but introduces storage overhead. Values shown assume a Batch Size of 4 on LLaMA-3-8B ($L = 32$, $h = 8$ KV heads, $d_k = 128$), yielding 64K values per token per sequence. INT4 quantization provides $4\times$ compression, making long-context high-throughput inference feasible.

For large context windows ($T \geq 10^5$), the KV cache can exceed GPU high-bandwidth memory (HBM) capacity—typically 40–80 GB on datacenter accelerators. This motivates quantization: an element-wise discretization map $\Pi : \mathbb{R} \to \mathcal{Q}$ that projects each cache entry onto a finite set $\mathcal{Q}$ of low-precision levels (e.g., $|\mathcal{Q}| = 16$ for 4-bit integers) (16). As shown in Table 1, INT4 quantization reduces storage by $4\times$, keeping the memory footprint tractable even at high batch sizes.

2.3. Algebraic Instability of Discretized Storage

Memory constraints require cache values to be discretized through an element-wise mapping $\Pi : \mathbb{R} \to \mathcal{S}$, where $\mathcal{S} \subset \mathbb{Z}$ is a finite set of representable integers (e.g., $\mathcal{S} = \{0, 1, \ldots, 15\}$ for INT4). This creates a mixed algebraic setting: the query vector $\mathbf{q} \in \mathbb{R}^d$ remains in continuous precision, while the cached matrices K and V are discrete approximations.

This discretization introduces two distinct sources of error:

1. **Quantization Error:** A bounded, deterministic error $\|\mathbf{x} - \Pi(\mathbf{x})\|$, typically modeled as uniform additive noise with magnitude at most half the quantization step size.

2. **Bit-Flip Errors:** Stochastic perturbations to the binary representation of stored integers, arising from DRAM soft errors, voltage noise in low-power regimes, or approximation in emerging memory technologies.

While quantization error is bounded and well-characterized, bit-flip errors pose a fundamentally different challenge. The storage channel is modeled as a Binary Symmetric Channel (BSC) with bit error rate (BER) p: each bit of a stored integer independently flips with probability p.

2.3.1. The Metric Mismatch

The core instability arises from a *metric mismatch* between the storage domain and the semantic domain. In storage, errors are naturally measured by *Hamming distance*—the number of differing bits—where a single bit flip has unit cost (26). However, when binary strings are interpreted as integers via positional encoding, bit positions carry exponentially different weights.

Example 1 (INT4 Metric Mismatch)**.** Consider a 4-bit unsigned integer with value $v = \sum_{i=0}^{3} b_i \cdot 2^i$. A flip in bit i changes the value by $\pm 2^i$:

- LSB flip ($i = 0$): $\Delta v = \pm 1$ (6% of range)
- MSB flip ($i = 3$): $\Delta v = \pm 8$ (50% of range)

A single-bit error in Hamming space (distance 1) can induce a perturbation spanning half the quantization range in Euclidean space.

More generally, for a B-bit integer representation, the ratio between maximum and minimum single-bit perturbations is 2^{B-1}. This exponential disparity means that uniform errors in Hamming distance produce highly non-uniform errors in Euclidean distance. A cache entry corrupted by a high-order bit flip becomes an *outlier* whose magnitude may exceed all legitimate values.

2.3.2. Propagation Through Softmax

This perturbation propagates catastrophically through the attention mechanism. Let $\tilde{\mathbf{k}} = \mathbf{k} + \mathbf{e}$ denote a corrupted key vector, where $\mathbf{e}$ is sparse (few corrupted entries) but potentially large in magnitude. The perturbed attention score becomes:

$$\tilde{s} = \langle \mathbf{q}, \tilde{\mathbf{k}} \rangle = \underbrace{\langle \mathbf{q}, \mathbf{k} \rangle}_{\text{true score}} + \underbrace{\langle \mathbf{q}, \mathbf{e} \rangle}_{\text{error term}}. \tag{6}$$

When $\|\mathbf{e}\|$ is large, the error term can dominate, producing an anomalously high (or low) score for the corrupted position.

The softmax function's exponential form amplifies this distortion. If the corrupted score $\tilde{s}_i$ exceeds other scores by margin δ, the attention weight becomes:

$$\alpha_i = \frac{\exp(\tilde{s}_i)}{\sum_j \exp(s_j)} \approx \frac{\exp(\tilde{s}_i)}{\exp(\tilde{s}_i)} = 1 \quad \text{as } \delta \to \infty. \tag{7}$$

The result is an approximately one-hot distribution concentrating nearly all probability mass on the corrupted entry. The attention mechanism then retrieves the corresponding value $\mathbf{v}_i$—which may be entirely irrelevant to the query—while ignoring the valid context.

This failure mode is catastrophic rather than graceful. A single corrupted cache entry can derail generation for an entire sequence, causing the model to produce repetitive or incoherent text. In the experiments described in Section 6, unprotected INT4 caches at BER $= 10^{-2}$ exhibit perplexity exceeding $1000\times$ the baseline, with 100% of samples experiencing catastrophic failure.

3. Related Work

3.1. KV Cache Compression

The memory bottleneck imposed by KV caches has motivated substantial work on compression techniques. Quantization-based approaches reduce the bit-width of cached values: KVQuant (16) applies per-channel quantization with outlier handling to achieve 3-bit keys and 2-bit values; KIVI (25) introduces asymmetric quantization treating keys and values differently based on their distributional properties; and Atom (37) combines mixed-precision quantization with fine-grained reordering. These methods focus on minimizing quantization error under the assumption of error-free storage.

Sparsity and eviction strategies reduce cache size by selectively retaining entries. H_2O (36) identifies "heavy hitter" tokens that accumulate disproportionate attention mass and evicts low-value entries. StreamingLLM (35) observes the "attention sink" phenomenon where initial tokens serve as stabilizing anchors and proposes retaining a sliding window plus sink tokens. Scissorhands (24) exploits persistence of importance across layers to prune redundant cache entries.

Architectural modifications avoid materialized caches entirely. Multi-Query Attention (MQA) (30) and Grouped-Query Attention (GQA) (1) share key-value heads across query heads, reducing cache size by factors of h or h/g respectively. Linear attention variants (18) replace softmax with kernel approximations enabling $\mathcal{O}(1)$ recurrent state, though often at the cost of model quality.

This work is orthogonal to these approaches: it addresses the *integrity* of cached values rather than their *size*. ECC protection can be composed with any quantization scheme, adding resilience without modifying the compression strategy.

3.2. Quantization for Large Language Models

Post-training quantization (PTQ) enables deployment of large models without retraining. GPTQ (12) formulates weight quantization as a layer-wise reconstruction problem, using approximate second-order information for optimal rounding. AWQ (23) identifies activation-aware salient weights and applies per-channel scaling to protect them. SmoothQuant (34) migrates quantization difficulty from activations to weights via mathematically equivalent trans-

formations.

For activations and KV caches, the challenge differs: values are not known at quantization time and exhibit dynamic ranges that vary across sequences. LLM.int8() (10) addresses activation outliers through mixed-precision decomposition, processing outlier dimensions in FP16. This work provides key empirical evidence for the *outlier emergence* phenomenon in quantized transformers—the observation that certain dimensions develop systematically large magnitudes that dominate quantization error.

The present framework builds on the insight from (10) that low-precision representations are vulnerable to large-magnitude perturbations, but addresses *stochastic* bit-level errors rather than *deterministic* quantization artifacts.

3.3. Error-Correcting Codes in Computing Systems

Classical coding theory (26) provides algebraic tools for detecting and correcting errors in digital communication and storage. Hamming codes (14) achieve single-error correction (SEC) with minimal redundancy; extended Hamming codes add double-error detection (SECDED) at the cost of one additional parity bit. Golay codes (13) provide stronger protection—the binary Golay code $\mathcal{G}_{24}$ corrects up to 3 errors in 24 bits—and are used in deep-space communication where retransmission is infeasible.

In computer architecture, ECC memory (ECC DRAM) protects against soft errors caused by cosmic radiation and alpha particle strikes (2). Standard ECC DIMMs implement SECDED at the 64-bit word level, adding 8 parity bits per word. However, this protection operates at the memory controller level and is transparent to software—applications cannot leverage stronger codes or application-specific error handling.

Algorithm-Based Fault Tolerance (ABFT) (17) takes a different approach, embedding checksums directly into matrix computations. For matrix multiplication $C = AB$, ABFT encodes A and B with row/column checksums that propagate through the computation, enabling detection and correction of errors in C. Extensions protect LU factorization (9), iterative solvers (5), and FFTs (33).

This work applies classical block codes at the *value level* rather than the word or matrix level, protecting individual quantized cache entries with application-aware encoding that accounts for the semantic importance of different bit positions.

3.4. Fault Tolerance in Neural Networks

Neural network resilience to hardware faults has been studied primarily in the context of safety-critical deployment. Reagen et al. (29) characterize the vulnerability of DNN accelerators to soft errors, finding that certain layers and weight bits are disproportionately sensitive. Li et al. (22) systematically inject faults into CNN inference, observing that errors in later layers and in high-magnitude weights cause the most severe accuracy degradation.

Selective protection strategies allocate redundancy non-uniformly based on vulnerability analysis. Mahmoud et al. (27) protect only the most critical neurons identified through gradient-based sensitivity analysis. Chen et al. (4) propose range restriction to bound the effect of bit flips, clipping activations to valid ranges.

For transformer architectures specifically, attention mechanisms present unique vulnerabilities due to the softmax nonlinearity. Heo et al. (15) analyze fault propagation in vision transformers, finding that errors in attention logits are amplified more severely than errors in MLP layers. However, this line of work focuses on *weight* errors during inference rather than *activation* errors in cached values.

3.5. Efficient Inference Systems

Production LLM serving systems have driven innovations in memory management and batching. vLLM (19) introduces *PagedAttention*, managing KV cache memory in non-contiguous blocks analogous to virtual memory paging. This enables efficient memory sharing across requests and reduces fragmentation. The implementation presented here builds on PagedAttention's block-based cache organization, applying ECC encoding at the block level.

FlashAttention (7; 6) fuses attention computation into a single kernel pass, reducing memory bandwidth by avoiding materialization of the full attention matrix. FlashDecoding (8) extends this to the decode phase with parallelization across the KV cache sequence dimension. The ECC kernels follow a similar philosophy, fusing encode/decode operations with cache access to minimize overhead.

Speculative decoding (21; 3) addresses latency by drafting multiple tokens with a smaller model and verifying with the target model. This is orthogonal to the present contribution but highlights the importance of cache integrity: speculative tokens that rely on corrupted cache entries would propagate errors through the verification step.

4. Theoretical Framework

Having established that the metric mismatch between Hamming and Euclidean spaces destabilizes the attention mechanism, this section formulates a protection strategy. The integrity of the cached matrices K and V is framed as a linear algebra problem over the finite field $\mathbb{F}_2$ (26).

BER (p)	Frequency	Expected Errors per 8-bit word	Regime
10^{-4}	1 in 10^4 bits	8×10^{-4}	Sparse
10^{-3}	1 in 10^3 bits	8×10^{-3}	Moderate
10^{-2}	1 in 10^2 bits	8×10^{-2}	Dense

Table 2. **Bit Error Rate Regimes.** Expected errors per 8-bit codeword under the BSC model. At $p = 10^{-2}$, approximately 1 in 12 codewords experiences at least one bit flip, challenging the correction capacity of single-error-correcting codes.

4.1. Stochastic Error Model over $\mathbb{F}_2$

The storage interface is modeled as a *Binary Symmetric Channel* (BSC). Let quantized integer values be represented as vectors in the finite-dimensional vector space $\mathbb{F}_2^k$, where k is the number of bits per value (e.g., $k = 4$ for INT4).

To facilitate error correction, a linear encoding map $\phi : \mathbb{F}_2^k \to \mathbb{F}_2^n$ is defined with $n > k$. Let $\mathbf{c} \in \mathbb{F}_2^n$ be the stored codeword. The retrieval process through a noisy channel yields:

$$\mathbf{r} = \mathbf{c} \oplus \mathbf{e}, \tag{8}$$

where $\oplus$ denotes addition in $\mathbb{F}_2$ (equivalently, bitwise XOR), and $\mathbf{e} \in \mathbb{F}_2^n$ is a stochastic error vector.

The components of $\mathbf{e} = [e_1, \ldots, e_n]$ are modeled as independent Bernoulli random variables with bit error rate (BER) p:

$$\Pr(e_i = 1) = p, \quad \Pr(e_i = 0) = 1 - p. \tag{9}$$

The expected Hamming weight of the error vector is $\mathbb{E}[w_H(\mathbf{e})] = np$. For an 8-bit codeword at BER $p = 10^{-2}$, this yields an expected 0.08 errors per codeword, or approximately 1 error per 12 codewords.

Table 2 summarizes the three BER regimes analyzed in this work. The dense regime ($p = 10^{-2}$) is particularly significant: the high error density challenges the correction capacity of standard codes, motivating the stronger constructions developed in Section 5.

4.2. Subspace Embeddings via Linear Block Codes

The protection mechanism restricts valid representations to a k-dimensional linear subspace $\mathcal{C} \subset \mathbb{F}_2^n$, called a *linear code*. By embedding k-bit data into an n-bit space with $n > k$, the encoding introduces $n - k$ bits of redundancy. This redundancy creates geometric separation between valid codewords, enabling detection and correction of errors (26).

The construction relies on two dual characterizations of $\mathcal{C}$: as the *image* of a generator matrix (for encoding) and as the *kernel* of a parity-check matrix (for decoding).

4.2.1. Generator Matrix and Encoding

Encoding is defined by an injective linear map $\phi : \mathbb{F}_2^k \to \mathbb{F}_2^n$ represented by a *generator matrix* $G \in \mathbb{F}_2^{k \times n}$ of full row rank:

$$\mathcal{C} = \mathrm{Im}(G) = \{\mathbf{d}G \mid \mathbf{d} \in \mathbb{F}_2^k\}. \tag{10}$$

For data vector $\mathbf{d}$, the codeword is $\mathbf{c} = \mathbf{d}G$.

Systematic generator matrices of the form $G = [I_k \mid P]$ are employed, where I_k is the $k \times k$ identity and $P \in \mathbb{F}_2^{k \times (n-k)}$ specifies the parity computation. This form preserves data bits in the first k positions of the codeword:

$$\mathbf{c} = \mathbf{d}G = [\mathbf{d} \mid \mathbf{d}P] = [\underbrace{d_1, \ldots, d_k}_{\text{data}} \mid \underbrace{p_1, \ldots, p_{n-k}}_{\text{parity}}]. \tag{11}$$

This property is critical for latency: in the absence of errors, the data can be read directly without decoding logic.

4.2.2. Parity-Check Matrix and the Kernel Characterization

For decoding, it is more efficient to characterize $\mathcal{C}$ as the kernel of a *parity-check matrix* $H \in \mathbb{F}_2^{(n-k) \times n}$:

$$\mathcal{C} = \ker(H) = \{\mathbf{x} \in \mathbb{F}_2^n \mid H\mathbf{x}^\top = \mathbf{0}\}. \tag{12}$$

For a systematic generator $G = [I_k \mid P]$, the corresponding parity-check matrix is:

$$H = [P^\top \mid I_{n-k}]. \tag{13}$$

The duality condition $GH^\top = \mathbf{0}$ ensures that all codewords satisfy the parity constraints: $\mathbf{c} \in \mathcal{C} \Rightarrow H\mathbf{c}^\top = \mathbf{0}$.

4.2.3. Minimum Distance and Correction Capability

The error-correction capability of a code is determined by the geometric separation between codewords.

Definition 4.1 (Minimum Distance)**.** The **minimum distance** of a linear code $\mathcal{C}$ is:

$$d_{\min} = \min_{\mathbf{c}_1 \neq \mathbf{c}_2 \in \mathcal{C}} w_H(\mathbf{c}_1 \oplus \mathbf{c}_2) = \min_{\mathbf{c} \in \mathcal{C}, \mathbf{c} \neq \mathbf{0}} w_H(\mathbf{c}), \tag{14}$$

where the second equality follows from linearity.

A code with minimum distance d can:

- **Detect** all error patterns of weight $\leq d - 1$.
- **Correct** all error patterns of weight $\leq t = \lfloor (d-1)/2 \rfloor$.

4.2.4. Syndrome Decoding

Given a received vector $\mathbf{r} = \mathbf{c} \oplus \mathbf{e}$, the decoder must recover $\mathbf{c}$ (or equivalently, identify $\mathbf{e}$). This is accomplished via the *syndrome*.

Definition 4.2 (Syndrome)**.** For parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ and received vector $\mathbf{r} \in \mathbb{F}_2^n$, the **syndrome** is:

$$\mathbf{z} = H\mathbf{r}^\top \in \mathbb{F}_2^{n-k}. \tag{15}$$

Since $\mathbf{c} \in \ker(H)$, we have $\mathbf{z} = H(\mathbf{c} \oplus \mathbf{e})^\top = H\mathbf{e}^\top$. The syndrome depends *only on the error pattern*, not the transmitted data.

For codes with small syndrome spaces (e.g., $2^3 = 8$ or $2^{12} = 4096$), the mapping from syndrome $\mathbf{z}$ to the most likely error pattern $\hat{\mathbf{e}}$ (the coset leader) can be precomputed in a lookup table. The corrected codeword is $\hat{\mathbf{c}} = \mathbf{r} \oplus \hat{\mathbf{e}}$.

4.3. Hierarchy of Algebraic Protection

This framework is instantiated with three specific codes of increasing strength:

1. **Hamming(7,4) [SEC]:** A perfect code with $d_{\min} = 3$. It embeds $k = 4$ data bits into $n = 7$ bits. It corrects any single-bit error ($t = 1$) but inevitably miscorrects double errors as single errors.

2. **Extended Hamming(8,4) [SECDED]:** Adds a parity bit to Hamming(7,4), increasing distance to $d_{\min} = 4$. It corrects single errors and *detects* double errors. This detection property is the foundation of the interpolation strategy presented here.

3. **Binary Golay(24,12):** A perfect code with $d_{\min} = 8$, capable of correcting up to $t = 3$ errors. It provides the strongest protection but requires significantly larger lookup tables.

5. Specific Code Constructions

Having established the general framework of subspace embeddings, specific code constructions are analyzed in this section. The central trade-off is between error-correction capability (determined by minimum distance $d_{\min}$) and storage overhead (determined by the rate $R = k/n$). Three codes of increasing strength are presented: Hamming(7,4) for single-error correction, extended Hamming(8,4) for single-error correction with double-error detection, and Golay(24,12) for triple-error correction.

5.1. The Hamming(7,4) Code

The Hamming(7,4) code is an $[n, k, d] = [7, 4, 3]$ linear code that embeds 4-bit data into 7-bit codewords, achieving single-error correction ($t = 1$) with 75% storage overhead (26).

5.1.1. Construction

The parity-check matrix $H \in \mathbb{F}_2^{3\times 7}$ is constructed by taking all $2^3 - 1 = 7$ non-zero vectors of $\mathbb{F}_2^3$ as columns. To ensure a systematic memory layout, the columns are permuted such that the identity matrix appears in the last 3 positions:

$$H = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \tag{16}$$

The corresponding systematic generator matrix $G = [I_4 \mid P] \in \mathbb{F}_2^{4\times 7}$, which satisfies $GH^\top = \mathbf{0}$, is:

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{17}$$

Encoding data $\mathbf{d} = [d_0, d_1, d_2, d_3]$ produces the codeword $\mathbf{c} = \mathbf{d}G = [d_0, d_1, d_2, d_3, p_0, p_1, p_2]$, where the parity bits are computed as:

$$p_0 = d_0 \oplus d_1 \oplus d_3, \tag{18}$$

$$p_1 = d_0 \oplus d_2 \oplus d_3, \tag{19}$$

$$p_2 = d_1 \oplus d_2 \oplus d_3. \tag{20}$$

5.1.2. Geometric Properties

The Hamming(7,4) code is a perfect code: the Hamming spheres of radius $t = 1$ centered at codewords partition $\mathbb{F}_2^7$ exactly.

Definition 5.1 (Perfect Code)**.** A code $\mathcal{C}$ with minimum distance $d = 2t + 1$ is perfect if every vector in the ambient space lies within distance t of exactly one codeword.

For Hamming(7,4):

- Minimum distance: $d_{\min} = 3$, enabling correction of $t = \lfloor (3-1)/2 \rfloor = 1$ error.
- Sphere-packing bound: Each radius-1 ball contains $\sum_{i=0}^{1} \binom{7}{i} = 1 + 7 = 8$ vectors. With $|\mathcal{C}| = 2^4 = 16$ codewords, total coverage is $16 \times 8 = 128 = 2^7 = |\mathbb{F}_2^7|$.

This property implies that every non-zero syndrome corresponds to a unique, correctable single-bit error pattern.

5.1.3. Syndrome Decoding

Decoding exploits the bijection between single-bit errors and columns of H. For an error $\mathbf{e}_j$ (a 1 in position j, zeros elsewhere):

$$\mathbf{z} = H\mathbf{e}_j^\top = \mathbf{h}_j \quad \text{(the } j\text{-th column of } H\text{)}. \tag{21}$$

Since all columns of H are distinct, the syndrome uniquely identifies the error position. Decoding reduces to a table lookup mapping $\mathbf{z} \mapsto j$.

Example 2 (Single-Error Correction)**.** Consider data $\mathbf{d} = [0, 1, 0, 0]$. Encoding yields:

$$\mathbf{c} = [0, 1, 0, 0] \cdot G = [0, 1, 0, 0, 1, 0, 1].$$

If bit index 3 (the 4th bit) is corrupted, the received vector is:

$$\mathbf{r} = [0, 1, 0, \mathbf{1}, 1, 0, 1].$$

The syndrome is:

$$\mathbf{z} = H\mathbf{r}^\top = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Comparison with Eq. 16 shows this matches column index 3 of H. Flipping this bit recovers the original codeword.

5.1.4. Limitation: Double-Error Miscorrection

A critical limitation of Hamming(7,4) is its behavior under double errors. With $d_{\min} = 3$, a weight-2 error $\mathbf{e}$ produces a syndrome $\mathbf{z} = H\mathbf{e}^\top = \mathbf{h}_i \oplus \mathbf{h}_j$. Since H contains all non-zero 3-bit vectors, this sum equals some other column $\mathbf{h}_k$. The decoder effectively "hallucinates" a single bit error at index k and flips it, resulting in a weight-3 error.

Example 3 (Miscorrection)**.** If errors occur at indices 0 and 1, the syndrome is:

$$\mathbf{z} = \mathbf{h}_0 \oplus \mathbf{h}_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \mathbf{h}_2.$$

The decoder incorrectly flips the bit at index 2. The corrected vector now differs from the original data by 3 bits.

This miscorrection behavior motivates the extended Hamming code, which detects double errors rather than miscorrecting them.

5.2. The Extended Hamming(8,4) Code

The miscorrection behavior of the Hamming(7,4) code under double errors motivates an extension. By appending a single overall parity bit, the minimum distance increases from $d = 3$ to $d = 4$, enabling SECDED: Single-Error Correction, Double-Error Detection.

5.2.1. Construction

The extended generator matrix $\bar{G} \in \mathbb{F}_2^{4\times 8}$ appends an overall parity column to the systematic Hamming(7,4) generator:

$$\bar{G} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}, \tag{22}$$

where the eighth column (index 7) ensures each codeword has even Hamming weight.

The extended parity-check matrix $\bar{H} \in \mathbb{F}_2^{4\times 8}$ is constructed by bordering the original matrix H:

$$\bar{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \tag{23}$$

Rows 0–2 compute the local Hamming syndrome; Row 3 computes the overall parity check. This construction satisfies the orthogonality condition $\bar{G}\bar{H}^\top = \mathbf{0}$.

5.2.2. SECDED Decoding Logic

For a received vector $\mathbf{r} \in \mathbb{F}_2^8$ (indexed $0 \ldots 7$), two distinct signals are computed:

$$\mathbf{z} = H_{7,4} \cdot \mathbf{r}_{[0:6]}^\top \in \mathbb{F}_2^3 \qquad \text{(Local Syndrome)} \tag{24}$$

$$p = \bigoplus_{i=0}^{7} r_i \in \mathbb{F}_2 \qquad \text{(Overall Parity)} \tag{25}$$

The pair $(\mathbf{z}, p)$ distinguishes four error cases:

Syndrome $\mathbf{z}$	Parity p	Diagnosis	Action
$\mathbf{0}$	0	No Error	Accept
$\neq \mathbf{0}$	1	Single Error ($w = 1$)	Correct via LUT
$\neq \mathbf{0}$	0	Double Error ($w = 2$)	Flag Erasure
$\mathbf{0}$	1	Parity Bit Error ($w = 1$)	Ignore (Data valid)

The key insight is that single-bit errors flip both the syndrome and the parity, while double-bit errors flip the syndrome but preserve parity (since $1 \oplus 1 = 0$).

When a double-bit error is detected, the decoder cannot uniquely identify which two bits are corrupted. Rather than miscorrecting (as Hamming(7,4) would), the decoder declares an erasure—the value is flagged as invalid.

5.3. Manifold-Aware Interpolation for Erasure Recovery

SECDED detection prevents corruption but creates missing data. Setting an erased entry to zero introduces systematic bias. Instead, the geometric structure of the KV cache sequence is exploited. The sequence of cached values $\{v_t\}_{t=1}^T$ is modeled as discrete samples from a smooth trajectory in $\mathbb{R}^d$.

5.3.1. Smoothness Assumption

The cache sequence is assumed to have bounded local variation. Consecutive entries satisfy:

$$\|v_t - v_{t-1}\|_2 \leq L\|v_{t-1}\|_2 \tag{26}$$

Strategy	Bias	Variance	Cost
Zero replacement	High (toward 0)	Low	$\mathcal{O}(1)$
Random noise	None	High	$\mathcal{O}(1)$
Keep corrupted	None	Extreme	$\mathcal{O}(1)$
Interpolation	Low	Low	$\mathcal{O}(d)$

Table 3. Erasure Handling Strategies. Linear interpolation minimizes both bias and variance by leveraging local smoothness, unlike naive zeroing or ignoring the error.

for some Lipschitz constant $L < 1$. This implies that neighboring values provide strong mutual information. Empirically, for both GPT-2 and LLaMA, the median relative change between consecutive keys is observed to be $< 10\%$ (Section 6).

5.3.2. Linear Interpolation Recovery

When position t is flagged as an erasure, $\hat{v}_t$ is estimated by minimizing the local curvature (second-order difference) of the reconstructed trajectory:

$$\hat{v}_t = \arg\min_{\mathbf{x}} \|(v_{t+1} - \mathbf{x}) - (\mathbf{x} - v_{t-1})\|_2^2. \tag{27}$$

Expanding the objective function $f(\mathbf{x})$:

$$f(\mathbf{x}) = \|(v_{t+1} + v_{t-1}) - 2\mathbf{x}\|_2^2. \tag{28}$$

Taking the gradient with respect to $\mathbf{x}$ and solving $\nabla_{\mathbf{x}} f = \mathbf{0}$:

$$-4(v_{t+1} + v_{t-1} - 2\mathbf{x}) = \mathbf{0} \quad \Rightarrow \quad \hat{v}_t = \frac{1}{2}(v_{t-1} + v_{t+1}). \tag{29}$$

The optimal estimate is the geometric midpoint of the neighbors. This approach:

- Preserves the scale of the sequence (unbiased).
- Requires only $\mathcal{O}(d)$ arithmetic operations.
- Is readily parallelizable within the attention kernel.

5.3.3. Boundary Handling

At sequence boundaries where a neighbor is undefined:

- Left boundary ($t = 0$): $\hat{v}_0 = v_1$ (Clamp to right neighbor).
- Right boundary ($t = T$): $\hat{v}_T = v_{T-1}$ (Clamp to left neighbor).

5.4. The Extended Binary Golay(24,12) Subspace

While Hamming codes provide efficient single-error correction, applications requiring stronger protection motivate codes with larger correction radius. The extended binary Golay code $\mathcal{G}_{24}$ is an $[n, k, d_{\min}] = [24, 12, 8]$ linear code—the unique perfect 3-error-correcting binary code of practical size.

5.4.1. Direct Sum Construction for INT4 Triplets

Unlike Hamming codes that encode a single INT4 value, Golay's 12-bit data capacity encodes three INT4 values as a triplet:

$$\mathbf{d} = [v_0 \mid v_1 \mid v_2] \in \mathbb{F}_2^{12}, \tag{30}$$

where each $v_i \in \{0, 1\}^4$ represents a 4-bit quantized cache value. This amortizes the 100% overhead (12 parity bits for 12 data bits) across three values, yielding an effective per-value overhead of 100%.

5.4.2. Algebraic Construction via Quadratic Residues

The Golay code derives from the Paley construction using quadratic residues modulo 11. Define the quadratic residue set:

$$\mathcal{Q}_{11} = \{x^2 \bmod 11 : x \in \{1, \ldots, 10\}\} = \{1, 3, 4, 5, 9\}.$$

The 12×12 matrix $\mathbf{B}$ is constructed with circulant structure based on these residues. The first row encodes membership in $\mathcal{Q}_{11}$, with subsequent rows formed by cyclic shifts, plus a final row to ensure the self-dual property:

$$\mathbf{B} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}. \tag{31}$$

5.4.3. Self-Orthogonality and Code Structure

The matrix $\mathbf{B}$ satisfies the remarkable self-dual property:

$$\mathbf{B}\mathbf{B}^\top \equiv \mathbf{I}_{12} \pmod 2. \tag{32}$$

This self-orthogonality ensures that the code is its own dual, enabling the systematic construction:

$$G_{24} = [\mathbf{I}_{12} \mid \mathbf{B}] \in \mathbb{F}_2^{12 \times 24}, \tag{33}$$

$$H_{24} = [\mathbf{B}^\top \mid \mathbf{I}_{12}] \in \mathbb{F}_2^{12 \times 24}. \tag{34}$$

The duality $G_{24}H_{24}^\top = \mathbf{0}$ follows from:

$$[\mathbf{I}_{12} \mid \mathbf{B}] \begin{bmatrix} \mathbf{B} \\ \mathbf{I}_{12} \end{bmatrix} = \mathbf{B} + \mathbf{B} = \mathbf{0} \pmod 2. \quad (35)$$

5.4.4. Combinatorial Properties and Minimum Distance

The Golay code achieves minimum distance $d_{\min} = 8$, providing:

- Error correction: $t = \lfloor (d_{\min} - 1)/2 \rfloor = 3$ bit errors
- Error detection: Up to $d_{\min} - 1 = 7$ bit errors
- Covering radius: Exactly 3 (every syndrome has a unique coset leader of weight ≤ 3)

The weight distribution of nonzero codewords forms a Steiner system $S(5, 8, 24)$: the 759 minimum-weight codewords (weight 8) partition the 24 coordinates into 8-element blocks such that every 5-element subset appears in exactly one block. This combinatorial structure underlies the code's optimality.

5.4.5. Syndrome Decoding via Precomputed Tables

Decoding employs a precomputed syndrome table of size $2^{12} = 4096$ entries. For received word $\mathbf{r} = \mathbf{c} + \mathbf{e}$, the 12-bit syndrome identifies the error pattern:

$$\mathbf{z} = H_{24}\mathbf{r}^\top = H_{24}\mathbf{e}^\top \in \mathbb{F}_2^{12}. \quad (36)$$

The table maps each syndrome to its unique correctable error pattern. The total number of correctable patterns is:

$$1 + \binom{24}{1} + \binom{24}{2} + \binom{24}{3} = 1 + 24 + 276 + 2024 = 2325. \quad (37)$$

Since Golay is a perfect code, these 2325 patterns produce exactly 2325 distinct syndromes, leaving $4096 - 2325 = 1771$ syndromes that indicate uncorrectable errors (≥ 4 bit flips). In the implementation, uncorrectable codewords return corrupted data with a flag, enabling downstream handling (e.g., discarding the value or using neighboring interpolation).

5.4.6. Comparison with Hamming Codes

Table 4 summarizes the trade-offs between the ECC options.

Golay's 3-error correction provides substantially stronger protection than Hamming at the cost of larger syndrome tables and increased decoding complexity. At high BER ($p = 10^{-2}$), where multi-bit errors within a codeword become likely, Golay maintains near-baseline perplexity while Hamming codes degrade measurably.

Table 4. ECC code comparison for INT4 KV cache protection.

Property	Hamming(7,4)	Hamming(8,4)	Golay(24,12)
Codeword length n	7	8	24
Data bits k	4	4	12
Minimum distance $d_{\min}$	3	4	8
Correction capability	1 bit	1 bit	3 bits
Detection capability	1 bit	2 bits	7 bits
Storage overhead	75%	100%	100%
Syndrome table size	8	8	4096
Values per codeword	1	1	3

5.4.7. Decoding Complexity

Unlike Hamming's $O(1)$ syndrome lookup in an 8-entry table, Golay requires a 4096-entry table lookup. However, this remains $O(1)$ per codeword—the table is precomputed at initialization and cached in GPU memory (16 KB for int32 entries). The dominant cost at inference time is the syndrome computation (12 parity checks over 24 bits), which the Triton implementation parallelizes efficiently across 256 elements per thread block.

6. Empirical Evaluation

We evaluate our ECC protection schemes on two transformer architectures: GPT-2 (124M parameters) and LLaMA-3.1-8B. Experiments use WikiText-2 test set with sliding window evaluation (max length 256, stride 128). Each configuration is evaluated across 3 random seeds with Monte Carlo fault injection at bit error rates (BER) $p \in \{0, 10^{-4}, 10^{-3}, 10^{-2}\}$.

6.1. Evaluation Metrics

Definition 6.1 (Perplexity)**.** Given a token sequence $X = (x_1, \ldots, x_T)$, the **Perplexity** (PPL) is defined as the exponential of the cross-entropy loss averaged over the sequence:

$$\text{PPL}(X) = \exp\left(-\frac{1}{T}\sum_{t=1}^{T} \ln P_\theta(x_t \mid x_{<t})\right).$$

Lower perplexity indicates better predictive performance. We additionally report **KL divergence** from clean outputs (measuring distribution shift) and **Top-5 accuracy** (fraction of positions where the true token appears in the model's top 5 predictions).

6.2. LLaMA-3.1-8B Results

Table 5 reports perplexity for LLaMA-3.1-8B under increasing bit-flip probability. The FP16 oracle baseline (no quantization, no bit flips) achieves PPL = 1.42. All INT4-quantized modes incur a small deterministic penalty from discretization, yielding PPL = 1.44 at $p = 0$.

At the critical regime $p = 10^{-2}$, where approximately 1%

Table 5. **LLaMA-3.1-8B Perplexity** under increasing BER. Values show mean ± 95% CI across 3 seeds.

Protection Mode	$p = 0$	$p = 10^{-4}$	$p = 10^{-3}$	$p = 10^{-2}$
FP16 (Oracle)	1.42	1.42	1.42	1.42
Hamming(7,4)	1.44	1.44	1.44	1.51±0.05
Hamming(8,4)	1.44	1.44	1.44	1.50±0.02
H(8,4)+Interp	**1.44**	**1.44**	**1.44**	**1.44±0.02**
Golay(24,12)	**1.44**	**1.44**	**1.44**	**1.44±0.01**

of stored bits are corrupted, both Hamming codes without interpolation show measurable degradation (PPL increases to 1.50–1.51). In contrast, Hamming(8,4) with interpolation and Golay(24,12) maintain baseline perplexity (1.44), demonstrating effective protection against high error rates.

6.3. GPT-2 Results

Table 6 shows analogous results for GPT-2. The smaller model exhibits similar patterns: Golay and H(8,4)+Interpolation maintain near-baseline performance even at $p = 10^{-2}$.

Table 6. **GPT-2 Perplexity** under increasing BER.

Protection Mode	$p = 0$	$p = 10^{-4}$	$p = 10^{-3}$	$p = 10^{-2}$
FP16 (Oracle)	1.78	1.78	1.78	1.78
Hamming(7,4)	1.77	1.77	1.78	1.86±0.03
Hamming(8,4)	1.77	1.77	1.77±0.02	1.92±0.18
H(8,4)+Interp	**1.77**	**1.77**	**1.78**	**1.77±0.03**
Golay(24,12)	**1.77**	**1.77**	**1.77**	**1.77±0.02**

6.4. Analysis of Algebraic Failures

A notable finding is that Hamming(8,4) without interpolation performs *worse* than Hamming(7,4) at $p = 10^{-2}$ (GPT-2: 1.92 vs. 1.86), despite having strictly stronger detection guarantees. This counterintuitive result is explained by examining how each scheme handles multi-bit errors.

Tables 7 and 8 report the error correction statistics for LLaMA-3.1-8B.

Table 7. **Single-Bit Errors Corrected** (LLaMA-3.1-8B). Number of codewords successfully corrected via syndrome decoding.

Protection	$p = 0$	$p = 10^{-4}$	$p = 10^{-3}$	$p = 10^{-2}$
Hamming(7,4)	0	57,848	577,332	5,606,766
Hamming(8,4)	0	58,154	575,589	5,394,530
Golay(24,12)	0	66,266	665,996	6,642,140

6.4.1. The Miscorrection vs. Erasure Trade-off

At $p = 10^{-2}$, Hamming(8,4) detects approximately 218,000 double-bit errors that it cannot correct. Without interpolation, these detected errors leave the corrupted data in place, producing large spurious attention scores. The cumulative effect of hundreds of thousands of such events degrades model quality.

Table 8. **Multi-Bit Errors Detected** (LLaMA-3.1-8B). Number of codewords flagged as uncorrectable (SECDED detection or Golay >3-bit events).

Protection	$p = 0$	$p = 10^{-4}$	$p = 10^{-3}$	$p = 10^{-2}$
Hamming(7,4)	0	0	0	0
Hamming(8,4)	0	47	2,340	218,139
Golay(24,12)	0	0	0	2,523

In contrast, Hamming(7,4) has no detection capability for double-bit errors. When two bits flip within a codeword, the decoder *miscorrects* to an incorrect but typically nearby codeword. While algebraically wrong, miscorrection often produces a value closer to the original than leaving corrupted data unchanged, explaining its relatively better performance.

This analysis motivates the interpolation strategy: when Hamming(8,4) detects an uncorrectable error, rather than preserving corrupted data, we reconstruct the value using temporal neighbors.

6.5. Validation of the Smoothness Hypothesis

The interpolation strategy tests our smoothness assumption: if cached state vectors vary smoothly across time, detected erasures can be approximated using neighboring vectors. Table 9 reports KL divergence from clean outputs, measuring distribution shift.

Table 9. **KL Divergence from Clean Outputs** (nats, LLaMA-3.1-8B). Lower values indicate outputs closer to the uncorrupted baseline.

Protection	$p = 0$	$p = 10^{-4}$	$p = 10^{-3}$	$p = 10^{-2}$
FP16 (Oracle)	0.000	0.000	0.000	0.000
Hamming(7,4)	0.013	0.013	0.014	0.073±0.002
Hamming(8,4)	0.013	0.013	0.013	0.060±0.022
H(8,4)+Interp	**0.013**	**0.013**	**0.014**	**0.019±0.004**
Golay(24,12)	**0.013**	**0.013**	**0.013**	**0.014±0.001**

At $p = 10^{-2}$, Hamming(7,4) and Hamming(8,4) without interpolation show substantial distribution shift (KL = 0.060–0.073). With interpolation enabled, KL divergence drops to 0.019, a 3× reduction. Golay achieves the lowest KL divergence (0.014), nearly matching the quantization-only baseline. This confirms that the KV cache exhibits sufficient temporal smoothness to support effective interpolation-based recovery.

6.6. Top-5 Prediction Accuracy

Table 10 reports the fraction of positions where the correct next token appears in the model's top-5 predictions, measuring prediction confidence under corruption.

Table 10. **Top-5 Accuracy** (LLaMA-3.1-8B). Higher is better.

Protection	$p = 0$	$p = 10^{-4}$	$p = 10^{-3}$	$p = 10^{-2}$
FP16 (Oracle)	98.2%	98.2%	98.2%	98.2%
Hamming(7,4)	98.2%	98.2%	98.1%	97.1±0.2%
Hamming(8,4)	98.2%	98.2%	98.1%	97.2±0.5%
H(8,4)+Interp	98.2%	98.2%	98.2%	97.8±0.2%
Golay(24,12)	98.2%	98.2%	98.2%	98.1±0.2%

All protection modes maintain high top-5 accuracy, with Golay showing the smallest degradation at high BER (98.1% vs. 98.2% baseline).

6.7. Summary and Discussion

Table 11 summarizes performance at the critical regime $p = 10^{-2}$ for LLaMA-3.1-8B.

Table 11. **Performance Summary at** $p = 10^{-2}$ (LLaMA-3.1-8B).

Metric	**FP16**	**Ham(7,4)**	**Ham(8,4)**	**H(8,4)+Interp**	**Golay**
PPL	1.42	1.51	1.50	**1.44**	**1.44**
ΔPPL	—	+6.3%	+5.6%	**+1.4%**	**+1.4%**
KL Divergence	0.00	0.073	0.060	**0.019**	**0.014**
Top-5 Accuracy	98.2%	97.1%	97.2%	97.8%	**98.1%**
Errors Corrected	—	5.6M	5.4M	5.4M	6.6M
Errors Detected	—	0	218K	218K	2.5K
Storage (bits/value)	16	7	8	8	8

7. Limitations and Future Work

7.1. Limitations

Prototype Implementation in Triton. Our ECC kernels are implemented in Triton (31), a Python-based DSL for GPU programming that provides portable performance across GPU architectures. While Triton enables rapid prototyping and achieves reasonable throughput, it does not match the performance of hand-optimized CUDA C++ implementations. Production inference systems such as vLLM (20), TensorRT-LLM, and SGLang use highly optimized CUDA kernels (typically distributed as `.cu`/`.cuh` files) that exploit architecture-specific features including warp-level primitives, shared memory tiling, and asynchronous memory operations. Our Triton implementation serves as a proof of concept demonstrating the feasibility and effectiveness of ECC protection, but production deployment would require reimplementation in CUDA C++ to minimize latency overhead.

No Integration with FlashAttention. FlashAttention (7; 6) has become the de facto standard for memory-efficient attention computation, achieving significant speedups through IO-aware tiling and avoiding materialization of the full attention matrix. Our current implementation uses a reference attention kernel that does not incorporate FlashAttention's optimizations. Integrating ECC encode/decode operations into FlashAttention's fused kernel structure presents nontrivial engineering challenges: the tiled computation pattern requires careful placement of syndrome computation to avoid redundant memory accesses, and the online softmax algorithm must be adapted to handle detected errors within tiles. A production-ready solution would require modifying FlashAttention's CUDA kernels to incorporate ECC operations at appropriate synchronization points.

No Integration with PagedAttention. vLLM's PagedAttention (20) manages KV cache memory using virtual memory concepts, enabling efficient batching of requests with different sequence lengths through non-contiguous block allocation. Our block-based ECC storage is conceptually compatible with PagedAttention's paging mechanism, but our prototype does not integrate with vLLM's memory manager or batch scheduler. Production integration would require modifications to vLLM's `CacheEngine` and `BlockManager` classes to allocate ECC-encoded blocks and handle the modified memory layout. Additionally, PagedAttention's copy-on-write semantics for beam search and parallel sampling would need to account for ECC metadata.

Binary Symmetric Channel Model. Our fault injection assumes a Binary Symmetric Channel (BSC) where each bit flips independently with probability p. While this model is standard in coding theory and provides a useful abstraction, real memory errors may exhibit spatial or temporal correlation (e.g., multi-bit upsets from single particle strikes, rowhammer effects, or wear-out patterns in specific memory regions). Our evaluation does not capture these correlated failure modes, which could affect the relative performance of different ECC schemes.

Limited Model and Dataset Scope. Our evaluation covers two model scales (GPT-2 at 124M parameters and LLaMA-3.1-8B) on a single benchmark (WikiText-2). While these experiments demonstrate the effectiveness of our approach, broader validation across diverse model families (e.g., encoder-decoder architectures, mixture-of-experts models), larger scales (70B+ parameters), and varied tasks (summarization, code generation, long-context retrieval) would strengthen the generality claims.

Synthetic Error Injection. All experiments use software-based fault injection rather than hardware-induced errors.

While our Monte Carlo approach provides statistical rigor, it does not capture potential interactions between memory errors and other system-level effects such as ECC corrections in DRAM, GPU memory controller behavior, or thermal variation. Hardware-in-the-loop validation using techniques such as beam injection or voltage manipulation would provide stronger evidence of real-world applicability.

7.2. Future Work

CUDA C++ Implementation. A natural next step is reimplementing our ECC kernels in CUDA C++ for integration with production inference frameworks. Key optimizations would include:

- Warp-level syndrome computation using `__ballot_sync` for parallel parity calculation
- Shared memory caching of syndrome lookup tables to reduce global memory traffic
- Vectorized loads/stores (`float4`, `int4`) for coalesced memory access
- Kernel fusion to amortize launch overhead across encode, attention, and decode phases

We estimate that a well-optimized CUDA implementation could reduce the ECC overhead from our current Triton baseline by 2–3$\times$.

FlashAttention Integration. Integrating ECC protection into FlashAttention requires inserting decode operations before K/V tile loads and encode operations after KV cache updates. The key challenge is maintaining FlashAttention's memory efficiency: ECC operations should occur within the existing tiling structure without requiring additional global memory round-trips. One approach is to decode ECC codewords into shared memory during the K/V load phase, perform attention computation on decoded values, and re-encode only for newly generated tokens. This would add approximately $O(B \cdot d)$ shared memory overhead per tile for storing decoded values, where B is the tile size and d is the head dimension.

vLLM and SGLang Integration. Production deployment requires integration with serving frameworks. For vLLM, this involves:

- Modifying `CacheEngine` to allocate ECC-encoded cache blocks
- Extending `BlockManager` to track ECC metadata and error statistics
- Adapting the PagedAttention CUDA kernel to perform inline ECC operations
- Exposing ECC configuration through the API for per-request or per-model settings

Similar modifications would enable integration with SGLang's RadixAttention and other emerging inference frameworks.

Adaptive ECC Selection. Our current approach uses a fixed ECC scheme throughout inference. An adaptive strategy could select protection strength based on runtime conditions:

- **Layer-wise adaptation**: Earlier layers may tolerate more errors than later layers due to the compositional nature of transformer computation
- **Error rate monitoring**: If detected error rates exceed a threshold, dynamically switch from Hamming to Golay protection
- **Criticality-aware protection**: Protect attention heads with higher gradient magnitudes more strongly, using techniques from neural network pruning literature

Hardware Validation. Validating our approach on actual hardware with induced memory errors would strengthen the practical relevance. Possible approaches include:

- Controlled voltage reduction to increase soft error rates
- Row-hammer attacks to induce targeted bit flips

Extended Quantization Support. Our current implementation focuses on INT4 quantization. Extending to other formats would broaden applicability:

- **FP8 (E4M3/E5M2)**: The 8-bit format used in modern inference accelerators; ECC could protect the exponent bits more strongly than mantissa bits based on sensitivity analysis
- **Mixed-precision**: Protect keys and values at different precision levels, as prior work suggests values may be more sensitive to quantization
- **Sub-4-bit quantization**: Emerging INT3 and INT2 schemes could benefit even more from ECC protection due to reduced representation capacity

Long-Context and Multi-GPU Scaling. As context windows extend to millions of tokens, KV cache memory becomes increasingly critical. Future work should evaluate ECC overhead at extreme sequence lengths and investigate:

- Distributed KV cache across multiple GPUs with per-device ECC

- Hierarchical protection with stronger codes for offloaded (CPU/SSD) cache
- Streaming decoding with incremental ECC updates

Theoretical Analysis of Error Propagation. While our empirical results demonstrate robustness, a theoretical analysis of how bit errors in KV cache propagate through attention layers would provide deeper understanding. Questions include:

- What is the expected perturbation in output logits as a function of BER and sequence length?
- Are certain attention patterns (e.g., local vs. global attention) more robust to cache corruption?
- Can we derive bounds on perplexity degradation under specific error models?

Such analysis could inform principled ECC selection criteria and enable error-aware model design.

References

[1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4895–4901, 2023.

[2] Robert C. Baumann. Radiation-induced soft errors in advanced semiconductor technologies. *IEEE Transactions on Device and Materials Reliability*, 5(3):305–316, 2005.

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

[4] Zitao Chen, Guanpeng Li, Karthik Pattabiraman, and Nathan DeBardeleben. Ranger: Range enforcement for DNNs using selective clipping. In *IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 373–384. IEEE, 2019.

[5] Zizhong Chen. Algorithm-based fault tolerance for fail-stop failures. In *IEEE International Parallel and Distributed Processing Symposium*, pages 1–8. IEEE, 2008.

[6] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[7] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359, 2022.

[8] Tri Dao, Daniel Haziza, Francisco Massa, and Grigory Sizov. Flash-decoding for long-context inference. Stanford CRFM Blog, 2023.

[9] J. T. Davies and Zizhong Chen. Algorithm-based fault tolerance for LU factorization. *Journal of Parallel and Distributed Computing*, 93:27–39, 1997.

[10] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems*, 35:30318–30332, 2022. Foundational work on outlier emergence in quantized LLMs.

[11] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Albert Alet, Amjad Raman, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. Technical Report covering Llama 3 and 3.1.

[12] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations (ICLR)*, 2023.

[13] Marcel J. E. Golay. Notes on digital coding. *Proceedings of the IRE*, 37:657, 1949.

[14] Richard W. Hamming. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, 1950.

[15] Suyeon Heo, Sunwoo Kim, Youngmin Kim, and Hyung-Sin Kim. Analyzing the resilience of vision transformers to bit-flip errors. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1–5. IEEE, 2023.

[16] Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*, 2024.

[17] Kuang-Hua Huang and Jacob A Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Transactions on Computers*, 100(6):518–528, 1984. Foundational work establishing the linear algebraic model for protecting matrix-vector products.

[18] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pages 5156–5165. PMLR, 2020.

[19] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E Gonzalez, Ion Stoica, and Hao Zhang. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

[20] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626. ACM, 2023. Alias for kwon2023efficient.

[21] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pages 19274–19286. PMLR, 2023.

[22] Guanpeng Li, Siva Kumar Sastry Hari, Michael Sullivan, Timothy Tsai, Karthik Pattabiraman, Joel Emer, and Stephen W. Keckler. Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 8:1–8:12. ACM, 2017.

[23] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*, volume 6, 2024. Best Paper Award.

[24] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems*, volume 36, pages 52342–52364, 2023.

[25] Zirui Liu, Jiayi Yuan, Hongye Jin, Zhong Shaochen, Zhuoran Xu, Vladimir Braverman, Beidi Chen, and Xia Ben Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

[26] Florence J MacWilliams and Neil JA Sloane. *The Theory of Error-Correcting Codes*. Elsevier, 1977. The definitive reference for the algebraic construction of Golay codes and the geometry of Hamming spaces.

[27] Abdulrahman Mahmoud, Neeraj Aggarwal, Alireza Esmaeildoust, Mehdi Hashemi, Sherief Reda, Alex Chiu, Nam Sung Kim, and Luc Moreau. HardNN: Feature map vulnerability evaluation in CNNs. In *IEEE International Symposium on Workload Characterization*, pages 171–182. IEEE, 2020.

[28] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[29] Brandon Reagen, Udit Gupta, Lillian Pentecost, Paul Whatmough, Sae Kyu Lee, Niamh Mulholland, David Brooks, and Gu-Yeon Wei. Ares: A framework for quantifying the resilience of deep neural networks. In *Proceedings of the 55th Annual Design Automation Conference*, pages 17:1–17:6. ACM, 2018.

[30] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

[31] Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19. ACM, 2019.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[33] Shyue-Ling Wang and Jing-Yang Jou. Algorithm-based fault tolerance for FFT networks. *IEEE Transactions on Computers*, 43(7):849–854, 1994.

[34] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pages 38087–38099. PMLR, 2023.

[35] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*, 2023.

[36] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 34661–34710, 2023.

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