ENCODING THE LLM VOCABULARY BOTTLENECK

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Models (LLMs) have achieved strong performance across a wide range of natural language processing tasks, but their increasing scale raises significant challenges in computational and memory efficiency. One of the major contributors to these costs is the decision layer (referred to as the *Softmax layer*), a fully connected projection followed by a Softmax activation. Its learned parameter count and associated optimizer-state count grow linearly with vocabulary size and can account for a substantial fraction of a smaller model's trainable state. In this work, we propose a framework based on Error Correcting Output Coding (ECOC), in which the standard Softmax layer is recovered as a special case corresponding to the identity coding, while more compact decision-boundary encodings are obtained by replacing this identity structure with alternative ECOC encodings, yielding an *ECOC-Softmax* decision layer. Experiments spanning three architectures and vocabularies of up to 114K tokens show that, with binary codewords of length 200, ECOC-Softmax keeps top-1 accuracy within around 2 percentage points of the best Softmax baseline while reducing the trainable output-layer parameter count by up to 571×, total trainable parameters by 18-23% and output-layer FLOPs per predicted token by 3.8-4.4×. Extending this, ECOC-Tying reduces total trainable parameters and associated optimizer state by 22-29% relative to tied Softmax.

1 INTRODUCTION

Large language models (LLMs) have become the backbone of modern natural language processing (NLP), enabling breakthroughs in machine translation, summarization, code generation, and multi-turn reasoning. Their success stems from scaling transformer architectures and training on massive corpora, allowing them to model complex, long-range linguistic dependencies (Wan et al., 2024; Touvron et al., 2023). Yet, this progress comes at a steep computational and memory cost, and as their adoption grows, the efficiency challenges of LLMs have become increasingly pronounced.

Two factors are key to this challenge: (i) **sequence length**: real-world applications, such as chain-of-thought reasoning, increasingly require handling longer contexts, which expands the key–value (KV) cache linearly, straining memory bandwidth during inference; (ii) less often acknowledged is **vocabulary expansion**: supporting diverse scripts, transliterations, and multilingual use cases demands vocabularies exceeding 100k tokens (Conneau et al., 2020; Bapna et al., 2022). The decision layer, implemented as a fully-connected projection with a Softmax activation, assigns one learned weight vector to each vocabulary entry, so its learned parameter count and associated optimizer-state values grow linearly with vocabulary size; in smaller models this layer alone can account for a large fraction of the trainable state. While advances such as FlashAttention (Dao et al., 2022), KV-cache compression (Anagnostidis et al., 2023), Speculative decoding (Leviathan et al., 2023), LoRA (Hu et al., 2022), quantization and pruning reduce the backbone's cost, they leave this component largely untouched. Thus, even highly optimized models remain hindered by its inefficiency.

We tackle precisely this flaw: the vocabulary bottleneck of the Softmax / decision layer. Our central idea is to replace this layer with a flexible and more general *Error Correcting Output Coding* (ECOC) framework. Originally adopted from information theory for multi-class learning, ECOC reframes multi-class classification problems as a collection of binary subproblems defined by a code matrix (Dietterich & Bakiri, 1995; Zor et al., 2010; 2016; Pujol et al., 2006; Ángel Bautista et al., 2012). Building on this foundation, we develop an *ECOC-Softmax* decision layer, in which the standard Softmax layer appears as a special ECOC design obtained using the identity code matrix, while alternative code matrices enable the encoding of different decision boundary formulations.

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

Under *minimal* ECOC designs, we show that the smallest number of binary subproblems needed to define a multi-class decision boundary reduces the number of output nodes from V in standard Softmax to just $\lceil \log_2 V \rceil$ for a problem with vocabulary of size V. We also extend the minimal design using random-bit augmentations, yielding robustness and accuracy trade-offs across model architectures and scales. Because the learned projection is sized by the code length L rather than the vocabulary size V, its trainable output-side parameter count and associated optimizer-state count no longer grow with vocabulary size. On C4 with a vocabulary of 114K tokens, a 200-dimensional ECOC-Softmax code reduces trainable output-layer parameters by up to $571\times$ and output-layer FLOPs per predicted token by $4.4\times$, while the top-1 accuracy gap to the best Softmax baseline remains within 2.2 percentage points. Overall, ECOC-Softmax reduces total trainable parameters and associated optimizer state by 18-23% relative to standard Softmax. Finally, ECOC-Tying shares the compact code-space factor between the input and output interfaces, removing the separate trainable vocabulary-sized input embedding table so that the entire trainable token interface scales with code length rather than vocabulary size. ECOC-Tying reduces total trainable parameters and associated optimizer state by 22-29% relative to tied Softmax. In brief, our contributions are:

- We introduce the ECOC-Softmax framework for the LLM decision layer, in which standard Softmax is the identity-code special case and compact codes (a minimal encoding and its random-bit extension) set the decision layer's size by code length rather than vocabulary size.
- We propose ECOC-Tying, a weight-tied variant that derives the input embedding from the same compact factor as the output projection, removing the separate vocabulary-sized embedding table.
- We provide an experimental study across three architectures and three pretraining corpora, mapping the accuracy-versus-cost trade-off across code lengths and identifying a configuration that trains stably across these settings.

2 Background

Efforts to make LLMs cheaper have largely focused on the transformer backbone, where most weights and compute reside. Quantization reduces weight and activation precision, from post-training INT8 (Dettmers et al., 2022; Yao et al., 2022) through low-bit pipelines (Xiao et al., 2023; Lin et al., 2024; Ashkboos et al., 2024; Lin et al., 2025; Zhao et al., 2024) toward sub-1-bit regimes (Gu et al., 2026), and pruning removes redundant structure, from block removal (Ma et al., 2023; Zhang et al., 2024) to structured and search-driven methods (An et al., 2024; Tang et al., 2025; Ling et al., 2024; Sieberling et al., 2025; Kang et al., 2025; Joo et al., 2025). These reduce backbone cost but leave the vocabulary-sized decision matrix untouched, which we turn to next.

In decoder-only LLMs, a forward pass over a sequence of S tokens with hidden dimension d and vocabulary size V has total complexity

$$\mathcal{O}(S^2 d + S d^2) + \underbrace{\mathcal{O}(SdV)}_{\text{decision layer}}, \tag{1}$$

where the first term captures self-attention and feedforward computation, and the highlighted second term is the output projection from hidden states to logits over V tokens.

Prior work reduces vocabulary-related cost in several distinct ways. Hierarchical and Adaptive Softmax replace a flat token distribution with conditional decisions. Hierarchical Softmax includes both binary-tree and class-based variants; the latter predicts a word's class first, the the word within that class (Goodman, 2001). Frequency-ordered classes are common in neural language models (Mikolov et al., 2011), and Adaptive Softmax likewise partitions the vocabulary by frequency while reducing the dimensionality of tail groups (Grave et al., 2017). These methods reduce the computation required for a training target, but retain a vocabulary-sized input embedding.

Other approaches change the tokens or candidate set rather than the output-layer parameterization. Token pruning and merging remove or combine input tokens (Bogoychev et al., 2024; Ushio et al., 2023; Bauwens & Delobelle, 2024; Lee & Hong, 2024); dynamic methods such as VocabTailor (Zhang et al., 2026) exploit lexical locality to load portions of the LM head on demand; and inference-time methods such as FR-Spec (Zhao et al., 2025) and CSV-Decode (Liu et al., 2026) restrict scoring to a selected sub-vocabulary. Embedding-space methods such as VQ-Logits (Shao et al., 2025) compress token representations. These approaches address complementary costs, but do not jointly replace the learned input and output token interfaces with a fixed codebook and a shared compact factor.

Error Correcting Output Coding (ECOC). In Section 3, we show that the V outputs produced by the standard Softmax layer for a vocabulary of size V can be interpreted as solutions to V *one-vs-all* sub-problems. Viewing the layer as a collection of binary sub-problems naturally motivates alternative decompositions that may yield more efficient or performant formulations than standard Softmax. A classic alternative is one-vs-one voting, which trains $V(V-1)/2$ pairwise classifiers and aggregates their outputs at test time (Hsu & Lin, 2002). This is where Error Correcting Output Coding (ECOC) comes into play. ECOC provides a unified framework in which a multi-class problem can be decomposed into binary classification problems in different ways, each capturing different class groupings and decision boundaries.

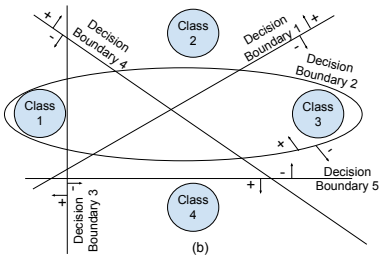
Adopted from error-correcting codes in information theory by Dietterich & Bakiri (1995), ECOC is a multi-class ensemble strategy that decomposes a V-class task into L binary sub-tasks governed by a code matrix $M \in \{0,1\}^{V \times L}$. Each column j of the code matrix (*codebook*) partitions the classes into a positive group, whose entries are 1, and a negative group, whose entries are 0. Since each column is a vector of $\{0,1\}$ entries, it is often referred to as a 'bit', i.e., a binary decision unit corresponding to one node in the ensemble and defining the two-class problem used to train base classifier h_j. Each row M_i is the unique codeword for class c_i, with the closest codeword to the classifier outputs determining the winning class. The encoding matrix can implement one-vs-all, one-vs-one, or intermediate multi-class decompositions.

For the 4-class and 5-column (i.e. 5-bit) formulation in Fig. 1-(a), the five resulting binary tasks are $(+)\{c_1, c_2\}$ vs. $(-)\{c_3, c_4\}$, $(+)\{c_1, c_3\}$ vs. $(-)\{c_2, c_4\}$, $(+)\{c_1\}$ vs. $(-)\{c_2, c_3, c_4\}$, $(+)\{c_2, c_3\}$ vs. $(-)\{c_1, c_4\}$, and $(+)\{c_4\}$ vs. $(-)\{c_1, c_2, c_3\}$. Fig. 1-(b) shows the corresponding decision boundaries of the individual base classifiers $h_{1,2,3,4}$.

At inference time, the ECOC ensemble produces a vector $\tilde{u}(X) = [\tilde{u}_1(X), \ldots, \tilde{u}_L(X)]$, with $\tilde{u}_j(X) \in (0,1)$, which is compared with the predefined codewords to recover the most likely class. Formally, the predicted class is the one whose codeword is closest to $\tilde{u}(X)$: $c^\star = \text{argmin}_{i \in \{1,\ldots,V\}} \, dist(\tilde{u}(X), M_i)$. The chosen distance metric (e.g., Hamming, Cosine, Euclidean) determines how the binary outputs are combined into the final multi-class decision (Fig. 1-(a)). The minimum Hamming distance between any two codewords in Fig. 1-(a) is 2, allowing an error in any one bit to keep the correct prediction without triggering an assignment to an incorrect codeword.

	h_1	h_2	h_3	h_4	h_5
c_1	1	1	1	0	0
c_2	1	0	0	1	0
c_3	0	1	0	1	0
c_4	0	0	0	0	1

(a)



(b)

Figure 1: Example ECOC matrix and corresponding decision boundaries for a 4-class problem with $L = 5$ binary classifiers.

By simplifying the classification task into loosely coupled binary problems, ECOC improves generalization, expands margins, and reduces overfitting, especially in high-cardinality output spaces (Allwein et al., 2001; Bautista et al., 2010). Prior work using ECOC for robust deep classification includes Ahmed et al. (2021); Verma & Swami (2019); Song et al. (2021); Wang et al. (2023); Jang et al. (2025); Yu et al. (2023).

3 ECOC-Softmax for Boundary Encoding and Optimization

In this work, we attach the ECOC construction to the language model's decision layer. Let $X \in \mathbb{R}^d$ denote the final transformer hidden representation at a token position, i.e., the representation that would ordinarily be passed to the vocabulary-sized output projection. While standard Softmax maps X directly to V token logits, ECOC-Softmax first projects X into L ECOC coordinates, one corresponding to each column of the codebook, and uses the fixed codebook M to construct the V class scores. This replaces the learned V-dimensional decision layer with an L-dimensional learned projection, where L can be much smaller than V.

For the theoretical analysis, we use the signed $\{-1, 1\}$ representation, related to the practical $0/1$ representation by $x_\pm = 2x_{01} - 1$ for ease of derivation. Let $M \in \{-1,1\}^{V \times L}$, where row $m_i = (m_{i1}, \ldots, m_{iL})$ is the codeword assigned to class i. Each column ℓ defines a binary superclass

partition, $\mathcal{S}_\ell^+ = \{i : m_{i\ell} = 1\}$ versus $\mathcal{S}_\ell^- = \{i : m_{i\ell} = -1\}$, while row m_i gives the superclass-side assignments for class i. The ECOC projection is parameterized by $W \in \mathbb{R}^{L\times d}$ and $b \in \mathbb{R}^L$, giving $u(X) = WX+b$, with $u_\ell(X) = w_\ell^\top X+b_\ell$. Coordinate ℓ corresponds to column ℓ of M, and its zero level set $\mathcal{H}_\ell = \{X : u_\ell(X) = 0\} = \{X : w_\ell^\top X + b_\ell = 0\}$ defines the associated superclass-aligned latent boundary. Here w_ℓ is the normal vector of this boundary and b_ℓ determines its offset. Thus, M specifies the superclass partition structure, while W parameterizes the corresponding superclass-aligned latent boundaries. The scalar $u_\ell(X)$ is its signed latent score: $u_\ell(X) > 0$ supports the code 1 side and $u_\ell(X) < 0$ supports the -1 side. These latent boundaries are learned jointly through their contribution to the final multiclass scores.

Collecting the coordinates gives $u(X) = [u_1(X), \ldots, u_L(X)]^\top$, which describes the position of X relative to the collection of superclass-aligned latent boundaries and forms a superclass-aligned margin pattern. For class i, $m_{i\ell}u_\ell(X) > 0$ means that coordinate ℓ supports class i's code assignment. We normalize this pattern as $\widehat{u}(X) = u(X)/\|u(X)\|$, preserving its signs and direction while removing overall magnitude. Similarly, since $m_{i\ell} \in \{-1, 1\}$, $\|m_i\| = \sqrt{L}$ and $\widehat{m}_i = m_i/\sqrt{L}$. The score for class i is therefore

$$z_i(X) = \widehat{m}_i^\top \widehat{u}(X) = \cos\theta_i,$$

where θ_i is the angle between the observed superclass-aligned margin pattern and class i's codeword.

For classes i and j, their pairwise comparison boundary is the set of points X where $z_i(X) = z_j(X)$. Define the pairwise multiclass margin $\gamma_{ij}(X) = z_i(X) - z_j(X) = (\widehat{m}_i - \widehat{m}_j)^\top \widehat{u}(X)$. Thus, $\gamma_{ij} > 0$ favors i, $\gamma_{ij} < 0$ favors j, and its zero level set $\mathcal{B}_{ij} = \{X : \gamma_{ij}(X) = 0\}$ is the corresponding i-vs-j pairwise boundary. The quantity γ_{ij} is derived from the class scores and is not an additional learned component of the model. Using $\widehat{m}_i = m_i/\sqrt{L}$,

$$\gamma_{ij}(X) = \frac{1}{\sqrt{L}} \sum_{\ell=1}^{L} (m_{i\ell} - m_{j\ell})\widehat{u}_\ell(X) = \frac{2}{\sqrt{L}} \sum_{\ell\in\mathcal{D}_{ij}} m_{i\ell}\widehat{u}_\ell(X),$$

where $\mathcal{D}_{ij} = \{\ell : m_{i\ell} \neq m_{j\ell}\}$. Coordinates for which $m_{i\ell} = m_{j\ell}$ cancel, while those whose code assignments distinguish i and j contribute to their pairwise margin. Each term $m_{i\ell}\widehat{u}_\ell(X)$ is the normalized contribution of coordinate ℓ oriented toward class i's code assignment: positive values support i over j, while negative values support j over i. Hence, γ_{ij} measures the net pairwise support obtained by combining the superclass-aligned coordinates that distinguish the two classes. Class i is top-scoring where $z_i(X) \geq z_j(X)$, equivalently $\gamma_{ij}(X) \geq 0$, for every $j \neq i$. Its hard decision region is therefore

$$\mathcal{R}_i = \bigcap_{j\neq i} \{X : \gamma_{ij}(X) \geq 0\} = \bigcap_{j\neq i} \left\{ X : \sum_{\ell\in\mathcal{D}_{ij}} m_{i\ell}\widehat{u}_\ell(X) \geq 0 \right\},$$

where the positive factor $2/\sqrt{L}$ has been omitted. For a fixed j, summing over ℓ constructs the pairwise margin γ_{ij}; intersecting over j gives the region in which class i has at least as large a score as every competitor.

The corresponding soft prediction is $p_i(X) = e^{z_i(X)}/\sum_k e^{z_k(X)}$. Softmax preserves the score ordering, so every $X \in \mathcal{R}_i$ also satisfies $p_i(X) \geq p_j(X)$ for all $j \neq i$. Defining $\mathrm{LSE}(z_{-i}(X)) = \log\sum_{j\neq i} e^{z_j(X)}$ and $\Delta_i(X) = z_i(X) - \mathrm{LSE}(z_{-i}(X))$, we obtain $p_i(X) = \sigma(\Delta_i(X))$. Since $\gamma_{ij}(X) = z_i(X) - z_j(X)$,

$$\Delta_i(X) = -\log\sum_{j\neq i} e^{-\gamma_{ij}(X)}.$$

Thus, the same pairwise margins that characterize the hard multiclass geometry are softly aggregated into the one-vs-rest confidence margin Δ_i.

In summary, M and W together define the superclass-aligned latent geometry. The signed scores $u_\ell(X) = w_\ell^\top X + b_\ell$ describe the position of X relative to these boundaries and together form the margin pattern used to obtain the class scores $z_i(X)$. Their pairwise differences $\gamma_{ij} = z_i - z_j$ are the induced multiclass margins, with $\gamma_{ij} = 0$ defining the i-vs-j pairwise boundaries. These margins determine both the hard decision regions $\mathcal{R}_i$ and, through Δ_i, the Softmax probabilities $p_i(X)$.

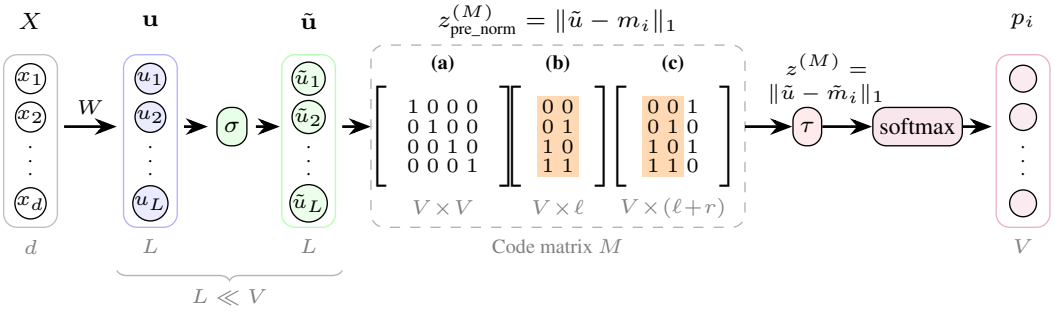


Figure 2: ECOC-Softmax decision layer architecture. The hidden state X is projected by a learned linear layer into L dimensions to form the embedding vector u, which passes through sigmoid σ to obtain $\tilde{u} \in [0,1]^L$. Then, $\tilde{u}$ is compared against a fixed binary codebook M via L_1 distance. The distances, normalized by temperature $\tau = L/L_{\text{ref}}$, are passed through Softmax to produce a probability distribution over V tokens. Three codebook designs are shown: (a) identity codes ($V \times V$, equivalent to one-vs-all), (b) Minimal ECOC ($V \times \ell$, $\ell = \lceil \log_2 V \rceil$), (c) MinRandom ($V \times (\ell + r)$).

Note that the analysis above uses the signed cosine formulation to expose this geometry, while the practical implementation depicted in Fig. 2 uses $0/1$ codes with L_1 decoding. Here, sigmoid is used for mapping the projected coordinates to the binary-code range $[0,1]^L$. Appendix E.1 relates this decoder to the signed formulation and shows that it preserves the superclass-aligned latent boundary structure. We use code-length temperature scaling and d-normalized initialization to control scale across code lengths and model widths, with the exact settings given in Section 4.1 and further analysis in Appendices E.1 and E.2. Using this framework, in our work, we consider two design choices: Minimal ECOC, which uses the smallest code capable of uniquely representing all V classes, and MinRandom ECOC, which adds redundant bits to increase codeword separation.

Minimal ECOC. For vocabulary size V, the shortest binary code assigning a unique codeword to every class has length $\ell = \lceil \log_2 V \rceil$. Minimal ECOC sets $L = \ell$, with each distinct binary string forming one row of the codebook (Fig. 2-(b)). This reduces the learned output projection from V dimensions in standard Softmax to $\lceil \log_2 V \rceil$ ECOC dimensions while still producing scores over all V tokens through the fixed codebook. For example, a vocabulary of 50,000 tokens requires only 16 learned output dimensions instead of 50,000.

MinRandom ECOC. MinRandom ECOC appends r random bits to each Minimal ECOC codeword (Fig. 2-(c)), increasing the code length from ℓ to $\ell + r$. Each bit is sampled independently from $\text{Bernoulli}(0.5)$, so the additional Hamming distance between any pair of codewords follows $\text{Binomial}(r, 0.5)$ and concentrates around $r/2$. The deterministic component has worst-case distance $d_{\min}^{\text{base}} = 1$, while the random component increases codeword separation. A union bound over all $\binom{V}{2}$ pairs gives

$$d_{\min} \geq 1 + \frac{r}{2} - \frac{\sqrt{r}}{2}\sqrt{2\ln\binom{V}{2}}. \tag{2}$$

For $V = 1{,}000$ and $r = 290$ (300 total bits), this predicts $d_{\min} \geq 102$; empirically $d_{\min} = 110$, corresponding to 54 correctable bit errors (full table in Appendix E.3).

ECOC Weight Tying. Classical weight tying uses one learned vocabulary matrix at both the input and output interfaces. ECOC-Tying instead shares the compact matrix $W \in \mathbb{R}^{L \times d}$ already used to parameterize the superclass-aligned latent coordinates, while the fixed codebook $M \in \{0,1\}^{V \times L}$ provides token identity. At the input,

$$E = MW, \qquad e_i = m_i W, \tag{3}$$

where $E \in \mathbb{R}^{V \times d}$ is the implicit embedding table. Thus, the same rows of W that parameterize the latent ECOC coordinates at the output also form the shared components from which token embeddings are constructed. At the output, the same W maps X into the L-dimensional code space as $u(X) = WX + b$, after which the predicted code is decoded using M. The fixed codebook therefore supplies token-specific structure at both interfaces while W is shared between representation and decision, eliminating separate learned $V \times d$ input and output matrices.

4 EXPERIMENTAL EVALUATION

We evaluate ECOC for autoregressive next-token prediction by pre-training decoder-only transformers from random initialization, avoiding the confound of Softmax-pretrained backbones whose representations are optimized for the standard output layer. Section 4.1 describes the setup, Section 4.2 evaluates performance, and Section 4.3 analyzes parameters, memory, FLOPs, and latency.

4.1 EXPERIMENTAL SETUP

Our experiments use two complementary settings. We begin with TinyStories (Eldan & Li, 2023) as a controlled diagnostic environment rather than as a proxy for general language modeling. Its restricted vocabulary and lower linguistic complexity allow all output-layer variants to be trained from scratch to a meaningful level of convergence, making broad matched evaluations across architectures, vocabulary sizes, and code lengths practical and reducing the risk that comparisons are dominated by severe undertraining. We train OPT-125M (Zhang et al., 2022), Qwen2-0.5B (Yang et al., 2024), and Mistral-7B (Jiang et al., 2023) with vocabularies of 1,000 tokens, 5,000 tokens, and each model's full effective TinyStories vocabulary (17.5K-32.2K tokens). These choices span approximately evenly spaced minimal code lengths of 10, 13, and 15 bits. We then analyse whether the resulting accuracy and code-length trends persist on more diverse corpora using **C4** (Raffel et al., 2020) with Qwen2-0.5B ($V = 114{,}179$) and **WikiText-103** (Merity et al., 2017) with OPT-125M ($V = 46{,}888$). Their larger vocabularies also provide more representative settings for the efficiency analysis. Within each setting, the ECOC variants and four baselines use the same transformer backbone, training data, and optimization; only the token input and output interfaces differ. Appendix A.3 provides the complete training and measurement configurations.

ECOC Matrix construction. The codebook has $\ell = \lceil \log_2 V \rceil$ minimal columns (the binary index of each token, guaranteeing uniqueness) plus r random columns for the MinRandom setting, where each entry is sampled independently and uniformly from $\{0, 1\}$ using a fixed seed for reproducibility. The total code length is $L = \ell + r$; the case $r = 0$ corresponds to the Minimal ECOC design. Error correction properties are analyzed in Section 3 and Appendix E.3; alternative codebook designs are compared in Appendix F.1.

Output-layer variants. We compare ECOC-Softmax and its tied variant, ECOC-Tying, with four baselines. Softmax learns separate vocabulary-sized matrices for the input embedding and output projection; tied Softmax reuses one matrix for both. Hierarchical Softmax (Goodman, 2001) replaces the flat distribution with a two-level class-based model that first predicts a token group and then the token within that group. Adaptive Softmax (Grave et al., 2017) partitions tokens by frequency, giving frequent tokens a full-dimensional head and infrequent ones a lower-dimensional projection. During training, both score only the target token's conditional path rather than all V tokens, which is why their per-target compute is lower. Both nonetheless retain a vocabulary-sized input embedding. Implementation specifics (group sizing, cutoffs, reduction factor) are in Appendix A.3.

Distance calculation. We use L1 distance as a soft relaxation of the Hamming distance used in classical ECOC decoding (Dietterich & Bakiri, 1995), for its interpretability in controlling error correction. It preserves the theoretical properties derived in Section 3, decomposes into efficient matrix operations (Appendix B.1), and provides a uniform gradient signal across codeword entries; alternative metrics (e.g. L2) give comparable accuracy (Appendix F.1).

Normalization. Following the sigmoid and temperature mechanisms introduced in Section 3, we set $\tau = L/L_{\text{ref}}$ with $L_{\text{ref}} = 100$. Sigmoid bounds each bit to $[0, 1]$, fixing the L1 distance range to $[0, L]$ regardless of logit magnitude and allowing temperature to follow a universal linear rule with code length rather than requiring per-configuration tuning. We initialize the projection weights with $\text{std} = 1/\sqrt{d}$ to make the initial logit scale approximately independent of hidden dimension. Together, these mechanisms provide a default configuration across architectures and code lengths.

4.2 PERFORMANCE EVALUATION

Our evaluation focuses on the trade-off between vocabulary-interface cost and next-token ranking quality, measured by top-1 and top-5 accuracy. For the sake of completeness, Appendix D.2 reports perplexity under the fixed temperature rule ($\tau = L/100$).

Table 1: Top-1 / top-5 accuracy (%) on TinyStories for ECOC-Softmax at representative code lengths (L) and four baselines, across three architectures and three vocabulary scales. Best values in each column are bolded. "min" is $L = \lceil \log_2 V \rceil$; entries marked "–" were not recorded. The complete set of code lengths is given in Appendix D.1 (Table 14).

	OPT-125M			**Qwen2-0.5B**			**Mistral-7B**		
Output layer / L	1K	5K	Full	1K	5K	Full	1K	5K	Full
ECOC-Softmax min	49.6 / 56.0	50.8 / 57.0	50.9 / 57.2	52.8 / 59.1	54.5 / 60.7	55.0 / 61.0	48.5 / 56.1	51.7 / 58.1	52.3 / 58.6
ECOC-Softmax L=100	61.5 / 88.6	62.4 / 84.8	62.3 / 83.8	62.7 / 90.7	64.7 / 87.6	64.8 / 86.6	61.4 / 88.4	62.5 / 84.8	62.5 / 84.2
ECOC-Softmax L=200	62.2 / 89.8	62.8 / 87.1	63.2 / 86.8	63.6 / 91.3	65.0 / 89.6	64.9 / 89.0	61.0 / 88.8	63.4 / 87.3	63.6 / 87.2
ECOC-Softmax L=300	62.3 / 89.9	63.3 / 87.9	63.3 / 87.5	63.8 / 91.4	65.1 / 89.9	64.9 / 89.6	61.5 / 89.2	64.6 / 88.7	63.9 / 88.0
ECOC-Softmax L=500	**62.4** / 90.0	63.5 / 88.1	63.3 / 87.7	64.0 / 91.5	65.2 / 90.1	65.0 / 89.8	61.8 / 89.4	64.1 / 88.4	64.3 / 88.5
Softmax	**62.4 / 90.1**	**64.0 / 88.6**	64.2 / 88.6	**64.5 / 91.8**	**66.0** / 90.6	66.4 / 90.7	**63.0 / 90.3**	**65.8 / 89.9**	**65.9 / 89.9**
Tied Softmax	**62.4** / 90.0	**64.0 / 88.6**	**64.8 / 89.0**	64.2 / 91.7	**66.0 / 90.7**	**66.6 / 90.9**	–	–	–
Adaptive Softmax	**62.4** / 90.0	63.8 / 88.3	64.2 / 88.4	64.2 / 91.6	65.9 / 90.5	66.2 / 90.5	–	–	–
Hierarchical Softmax	62.2 / 89.9	63.5 / 88.3	63.9 / 88.4	63.9 / 91.5	65.6 / 90.4	65.9 / 90.4	–	–	–

TinyStories evaluation. Table 1 reports results across three architectures (OPT-125M, Qwen2-0.5B, and Mistral-7B) and three vocabulary scales. Accuracy rises steeply over the first 50-100 bits beyond the minimal $\lceil \log_2 V \rceil$ code and then shows diminishing gains: from 300 to 1,000 bits, top-1 changes by at most 1.3 percentage points. At $L = 200$ and $L = 300$, ECOC is within 2 percentage points of Softmax in 16 of 18 configurations. The remaining two cases, both at $L = 200$ for Mistral-7B at the 5K and full vocabulary, have gaps of 2.4 and 2.3 points, respectively. The complete code-length sweep is reported in Appendix D.1. This pattern holds from $d = 768$ to $d = 4096$ and from 125M to 7B parameters.

The recorded baselines are closely grouped: relative to Softmax, the largest top-1 difference is 0.6 points for tied Softmax, 0.3 for Adaptive Softmax, and 0.6 for Hierarchical Softmax. Where all four baselines were recorded, ECOC at $L = 200$–300 is within 1.7 points of the best baseline. Given this agreement and the cost of extra 7B runs, we did not repeat the remaining baselines for Mistral-7B. Section 4.3 examines how these output layers differ in parameter storage, memory, and computation.

Table 2: Next-token accuracy (%) on C4 and WikiText-103. All arms were trained under identical conditions; N/A indicates that code length does not apply to baselines.

		C4 Qwen2-0.5B		**WikiText-103** OPT-125M	
Output layer	L	Top-1	Top-5	Top-1	Top-5
ECOC-Softmax	100	30.4	47.1	39.9	54.5
ECOC-Softmax	200	31.1	51.0	40.2	57.8
ECOC-Softmax	300	31.2	52.2	40.4	59.0
ECOC-Softmax	400	31.4	52.7	40.2	59.5
ECOC-Softmax	500	31.2	52.8	40.2	59.7
ECOC-Tying	200	31.4	51.2	39.8	57.4
ECOC-Tying	500	31.3	53.0	39.9	59.2
Softmax	N/A	33.1	55.8	**42.0**	**62.5**
Tied Softmax	N/A	**33.3**	**56.1**	**42.0**	62.4
Adaptive Softmax	N/A	32.9	55.6	41.9	62.0
Hierarchical Softmax	N/A	32.5	55.3	41.6	62.0

Full-vocabulary evaluation on C4 and WikiText-103. Table 2 tests if the TinyStories trend extends to full vocabularies on C4 and WikiText-103 (114K and 47K tokens), using held-out validation data.

For ECOC-Softmax, top-1 accuracy is largely stable between 200 and 400 bits on both corpora, consistent with the saturation observed on TinyStories. At $L = 200$, the gap between the reported ECOC variants and the best baseline is 1.8 and 2.2 percentage points in top-1 accuracy for the two corpora (Table 2). Increasing L beyond 300 yields no consistent top-1 gain for ECOC-Softmax, while top-5 continues to improve, indicating improvements in lower-ranked predictions without consistent changes to the top-1 prediction. The four baselines are tightly grouped (within 0.8 points in top-1). Together with the TinyStories results, these experiments support $L = 200$-300 as a common code-length range across the tested settings.

Appendix D.2 provides a complementary evaluation of the predictive distributions. Although ECOC preserves next-token rankings with modest degradation, its perplexity is higher than Softmax, particularly at full vocabulary, revealing a probabilistic-quality trade-off not captured by top-k accuracy. It must be noted that the development of probability estimation and calibration methods tailored to compact ECOC parameterization remains an open research direction as is beyond the present study. In our case, the fixed rule $\tau = L/100$ controls score scale across code lengths rather than fitting probability calibration, and we have not established how much of the observed gap post-hoc calibration could recover. The results support substantial vocabulary-interface compression with competitive ranking accuracy without establishing exact equivalent generation quality and leaving room for improving predictive log-loss.

Table 3: Training efficiency. Trainable token-interface parameters are in thousands and optimizer states in millions. An input side marked "derived" is computed as MW rather than stored. "Complete model" (millions) is the backbone plus trainable token interface, excluding the frozen $V \times L$ code matrix. Adaptive Softmax values (†) are frequency-weighted ranges at 90% head mass; Hierarchical Softmax values (‡) are nominal target-path counts. See Appendices B.2 and C.4.

	computed: architecture and AdamW						*measured: fixed setting*		
	trainable token-interface params			Complete	Optimizer	Output	Optimizer	Peak	Whole
Output layer	input side (K)	output side (K)	total (K)	model (M)	state (M)	GEMM (M/tok)	update (ms)	mem (MB)	step (ms)
C4, Qwen2-0.5B, $d \times V = 102{,}304K$, $d \times L = 179K$, effective batch 64									
Softmax	102,304	102,304	204,609	562.5	1125.0	613.8	60.4	18,020	2603.6
Tied Softmax	102,304	*shared*	102,304	460.2	920.4	613.8	45.0	17,199	2646.8
Adaptive Softmax	102,304	20,462	122,767	480.7	961.3	64.5–65.3†	45.8	11,125	2181.1
Hierarchical Softmax	102,304	102,607	204,912	562.8	1125.6	**3.6**‡	60.1	**5,615**	**1922.8**
ECOC-Softmax $L = 200$	102,304	**179**	102,484	460.4	920.8	92.4	44.8	11,663	2228.5
ECOC-Tying $L = 200$	*derived*	**179** *(shared)*	**179**	**358.1**	**716.2**	92.4	**30.0**	10,855	2208.9
WikiText-103, OPT-125M, $d \times V = 36{,}010K$, $d \times L = 154K$, effective batch 128									
Softmax	36,010	36,010	72,020	158.6	317.3	216.1	20.2	12,390	1189.1
Tied Softmax	36,010	*shared*	36,010	122.6	245.3	216.1	14.8	12,102	1186.4
Adaptive Softmax	36,010	10,579	46,589	133.2	266.4	42.3–42.9†	15.4	7,100	928.6
Hierarchical Softmax	36,010	36,177	72,187	158.8	317.6	**2.0**‡	20.0	**2,181**	**730.1**
ECOC-Softmax $L = 200$	36,010	**154**	36,164	122.8	245.6	38.4	15.0	7,544	974.2
ECOC-Tying $L = 200$	*derived*	**154** *(shared)*	**154**	**86.8**	**173.6**	38.4	**12.4**	7,271	970.1

4.3 Efficiency Analysis

Efficiency here has two separate parts: how many trainable parameters are associated with the input interface and decision layer, and how much memory and computation these components require during execution. Here, the *input interface* denotes the part of the architecture that maps token identities to the d-dimensional embeddings consumed by the transformer. Together, we refer to the input interface and decision layer as the *vocabulary interface*; the learned parameters associated with these components are referred to as the *learned token-interface* parameters.

We use untied Softmax as the reference for ECOC-Softmax and tied Softmax for ECOC-Tying. Adaptive and Hierarchical Softmax provide additional baselines that reduce training computation through target-dependent evaluation. The measurement protocol is detailed in Appendix A.3; FLOP derivations and execution-sensitivity results are given in Appendices B and C.

Training efficiency. Table 3 separates architectural savings (the trainable input-embedding and decision-layer parameters, and the optimizer state they induce) from measured training cost (optimizer-update time, peak memory, and whole-step time). Peak memory also includes activations, gradients, logits, and temporary workspaces, so it need not fall by the same rate as the trainable parameter count.

ECOC-Softmax reduces the output matrix from $d \times V$ to $d \times L$, approximately halving the trainable token-interface parameters. This lowers the complete-model parameter count by 18.2% on C4 (562.5M to 460.4M) and 22.6% on WikiText-103 (158.6M to 122.8M). ECOC-Tying further replaces the shared $V \times d$ matrix of tied Softmax with a shared trainable $L \times d$ factor, reducing the trainable token-interface parameters by V/L (571× on C4 and 234× on WikiText-103). With the backbone unchanged, this yields additional complete-model reductions of 22.2% on C4 (460.2M to 358.1M) and 29.2% on WikiText-103 (122.6M to 86.8M).

AdamW stores two optimizer values per learned parameter, so the smaller interface also cuts optimizer storage (Table 3). Relative to tied Softmax, ECOC-Tying lowers peak memory by 36.9% on C4 (17,199 to 10,855,MB) and 39.9% on WikiText-103 (12,102 to 7,271,MB), and whole-step time by 16.5% (2646.8 to 2208.9,ms) and 18.2% (1186.4 to 970.1,ms). Hierarchical Softmax attains the lowest peak memory and fastest step in both, because during training it scores only the target token's path when the label is known. These figures use unfused cross-entropy; fusing it improves the ECOC heads further, closing most of the gap to Hierarchical Softmax (Appendix C.4).

Inference efficiency. At autoregressive inference there are no gradients or optimizer values. Table 4 therefore reports the memory occupied by the input embedding, decision layer, and codebook; the arithmetic required to produce one set of vocabulary scores; and measured decode-head latency.

$B = 1$ is the single-stream setting, while $B = 512$ shows whether the advantage holds when many hidden states are scored together.

The arithmetic reduction follows directly from the decoder. Softmax's dominant output cost is $2dV$, whereas ECOC uses a $d \times L$ projection and an $L \times V$ codebook comparison, giving dominant cost $2dL + 2LV$. At $L = 200$, this reduces output FLOPs by $4.4\times$ on C4 and $3.8\times$ on WikiText-103.

ECOC-Tying reduces resident interface storage by the same factors relative to tied Softmax. These reductions are smaller than the $571\times$ and $234\times$ reductions in trainable token-interface parameters because the fixed codebook still occupies memory; ECOC-Softmax additionally retains the standard input embedding.

Table 4: Main inference-efficiency comparison at $L = 200$. Storage covers the input embedding, decision layer, and fixed codebook, not the whole model. The full batch-size sweep is reported in Appendix A.4; adaptive-softmax FLOPs are frequency-dependent.

Output layer	MB	MFLOP/tok	$B=1$	$B=512$
C4, Qwen2-0.5B				
Softmax	409.2	204.6	0.921	3.428
tied Softmax	204.6	204.6	0.874	3.516
Adaptive softmax	245.5	**40.9**	0.463	3.391
Hierarchical softmax	409.8	205.2	0.952	6.465
ECOC-Softmax	250.7	46.0	**0.165**	0.958
ECOC-Tying	**46.0**	46.0	**0.165**	**0.955**
WikiText-103, OPT-125M				
Softmax	144.0	72.0	0.338	0.603
tied Softmax	72.0	72.0	0.339	0.603
Adaptive softmax	93.2	21.2	0.437	1.064
Hierarchical softmax	144.4	72.4	0.412	1.831
ECOC-Softmax	91.1	**19.1**	0.134	0.333
ECOC-Tying	**19.1**	**19.1**	**0.127**	**0.329**

Computing the ECOC scores as a dense codebook matmul (Appendix C.4), ECOC-Tying is $5.3\times$/$2.7\times$ faster than tied Softmax at $B = 1$ and $3.7\times$/$1.8\times$ faster at $B = 512$ on C4/WikiText-103, respectively. In these measurements, both ECOC variants are faster than all baselines across the batch-size sweep (Appendix A.4). Hierarchical Softmax, the fastest head to train, loses that advantage at inference: it is slower to decode than ECOC and no faster than Softmax (Table 4). Hierarchical Softmax can also decode by scoring only its top-ranked clusters rather than the full vocabulary; this cuts its decode arithmetic but not its latency (Appendix C.4).

Summary. ECOC and baseline Softmax methods address different efficiency bottlenecks. ECOC compresses the learned vocabulary interface, reducing model parameters, optimizer storage, and peak training memory, while providing efficient full-vocabulary scoring and lower decode latency across all tested batch sizes. Adaptive and Hierarchical Softmax instead reduce target-dependent training work, with hierarchical Softmax achieving the lowest peak memory and shortest training step in our measurements. Thus, ECOC favors compact learned vocabulary state and efficient full-vocabulary decoding, while target-dependent softmax methods can be more efficient during training.

5 Conclusions

We propose ECOC-Softmax, a framework based on Error Correcting Output Codes (ECOC) that generalizes the standard Softmax output layer (Section 3): a trainable $d \times L$ projection and a fixed binary code matrix replace the dense $d \times V$ projection, reducing trainable output-layer parameters from $O(dV)$ to $O(dL)$ with $L \ll V$, and recovering Softmax as the special case $M = I$. Across three architectures on three corpora, the code-based decision layers reach top-1 accuracy within about 2 percentage points of the best Softmax baseline at $L = 200$–300. Relative to the standard $d \times V$ Softmax output layer (untied and tied), they also reduce output-layer FLOPs per token, and ECOC-Tying further lowers trainable parameters, optimizer state, and decode latency across batch sizes (Section 4.3).

More broadly, ECOC treats the output layer as a vocabulary-compression problem, trading accuracy against code length. This is particularly relevant for large multilingual or domain-specific vocabularies and memory-constrained deployment. Because ECOC changes the token-interface architecture rather than the backbone, it is complementary to quantization, pruning, gradient checkpointing, and related efficiency methods.

Limitations and future work. We evaluate next-token prediction on TinyStories, C4, and WikiText-103, but not downstream reasoning or knowledge benchmarks. Efficiency measurements use one NVIDIA L4 and sequence length 256, so absolute latency and memory may differ on other hardware or longer contexts; the large-batch decode behavior may also change with fused kernels, and the temperature rule $\tau = L/100$ is validated only up to 7B parameters. Future work includes structured or trainable codebooks, selecting L from a target accuracy, fused code-space kernels and bit-packed decoding, and evaluation at larger scales, longer contexts, and downstream or multimodal tasks.

Ethics statement

This work studies the parameterization of the output layer of language models and introduces no new data collection involving human subjects. All models are trained from scratch on publicly available corpora (TinyStories, C4, WikiText-103) under their respective licenses. The method changes only how vocabulary logits are computed and does not target any application with foreseeable harm beyond those already associated with general-purpose language models. We are not aware of any ethical concerns specific to this work.

Reproducibility statement

We describe the ECOC-Softmax framework and its codebook construction in Section 3 and Section 4.1, and the full training and measurement configuration in Appendix A.3 and Appendix A.2. Codebooks are generated from a fixed seed, as described in Section 4.1, and the FLOP and efficiency accounting conventions are given in Appendix B.2 and Appendix B.1. Complete per-configuration accuracy and perplexity results are reported in Appendix D.1 and Appendix D.2.

AI use statement

We used generative AI tools as general-purpose writing and coding assistants: to help edit and tighten the prose of this manuscript, and to assist with routine code and LaTeX formatting. We did not use generative AI to generate research ideas, experimental results, or data. All AI-assisted text and code were reviewed and verified by the authors, and we take full responsibility for the final content of this work, including all text, claims, and artifacts.

References

Sara Atito Ali Ahmed, Cemre Zor, Muhammad Awais, Berrin Yanikoglu, and Josef Kittler. Deep convolutional neural network ensembles using ecoc. *IEEE Access*, 9:86083–86095, 2021. doi: 10.1109/ACCESS.2021.3088717.

Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary: a unifying approach for margin classifiers. *J. Mach. Learn. Res.*, 1:113–141, January 2001. ISSN 1532-4435. doi: 10.1162/15324430152733133. URL https://doi.org/10.1162/15324430152733133.

Yongqi An, Xu Zhao, Tao Yu, Ming Tang, and Jinqiao Wang. Fluctuation-based adaptive structured pruning for large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 10865–10873, 2024.

Sotiris Anagnostidis, Dario Pavllo, Luca Biggio, Lorenzo Noci, Aurelien Lucchi, and Thomas Hofmann. Dynamic context pruning for efficient and interpretable autoregressive transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=uvdJgFFzby.

Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L. Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. QuaRot: Outlier-free 4-bit inference in rotated LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2404.00456.

Ankur Bapna, Isaac Caswell, Julia Kreutzer, Orhan Firat, Daan van Esch, Aditya Siddhant, Mengmeng Niu, Pallavi Baljekar, Xavier Garcia, Wolfgang Macherey, Theresa Breiner, Vera Axelrod, Jason Riesa, Yuan Cao, Mia Xu Chen, Klaus Macherey, Maxim Krikun, Pidong Wang, Alexander Gutkin, Apurva Shah, Yanping Huang, Zhifeng Chen, Yonghui Wu, and Macduff Hughes. Building machine translation systems for the next thousand languages, 2022. URL https://arxiv.org/abs/2205.03983.

Miguel A Bautista, Xavier Baró, Oriol Pujol, Petia Radeva, Jordi Vitrià, and Sergio Escalera. Compact evolutive design of error-correcting output codes. In *Proceedings of the Supervised and Unsupervised Methods and their Applications (SUEMA), European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2010.

Thomas Bauwens and Pieter Delobelle. BPE-knockout: Pruning pre-existing BPE tokenisers with backwards-compatible morphological semi-supervision. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.naacl-long.324/.

Nikolay Bogoychev, Pinzhen Chen, Barry Haddow, and Alexandra Birch. The ups and downs of large language model inference with vocabulary trimming by language heuristics. In *Proceedings of the Fifth Workshop on Insights from Negative Results in NLP*. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.insights-1.17/.

Raj Chandra Bose and Dwijendra K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960.

Alexis Conneau, Shijie Wu, Haoran Li, Luke Zettlemoyer, and Veselin Stoyanov. Emerging cross-lingual structure in pretrained language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 6022–6034, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.536. URL https://aclanthology.org/2020.acl-main.536/.

Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Re. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=H4DqfPSibmx.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. LLM.int8(): 8-bit matrix multiplication for transformers at scale. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2208.07339.

Terry G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995. doi: 10.1613/jair.105. URL https://doi.org/10.1613/jair.105.

Ronen Eldan and Yuanzhi Li. Tinystories: How small can language models be and still speak coherent english?, 2023. URL https://arxiv.org/abs/2305.07759.

Joshua Goodman. Classes for fast maximum entropy training. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pp. 561–564, 2001.

Édouard Grave, Armand Joulin, Moustapha Cissé, David Grangier, and Hervé Jégou. Efficient softmax approximation for gpus. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1302–1310. PMLR, 2017. URL https://proceedings.mlr.press/v70/grave17a.html.

Hao Gu, Lujun Li, Hao Wang, Lei Wang, Zheyu Wang, Bei Liu, Jiacheng Liu, Qiyuan Zhu, Sirui Han, and Yike Guo. BTC-LLM: Efficient sub-1-bit LLM quantization via learnable transformation and binary codebook. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2026. URL https://aclanthology.org/2026.acl-long.1066/.

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multi-class support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002. doi: 10.1109/72.991427. URL https://doi.org/10.1109/72.991427.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Wooram Jang, Woojin Hwang, Kezhong Jin, and Hosung Park. Bit flipping-based error correcting output code construction for adversarial robustness of neural networks. *ICT Express*, 11(2): 348–353, 2025. ISSN 2405-9595. doi: https://doi.org/10.1016/j.icte.2025.02.002. URL https://www.sciencedirect.com/science/article/pii/S2405959525000128.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Donghyeon Joo, Helya Hosseini, Ramyad Hadidi, and Bahar Asgari. Coruscant: Co-designing gpu kernel and sparse tensor core to advocate unstructured sparsity in efficient llm inference. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture*, MICRO '25, pp. 232–245, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400715730. doi: 10.1145/3725843.3756065. URL https://doi.org/10.1145/3725843.3756065.

Yuhan Kang, Yang Shi, Mei Wen, Jun He, Jianchao Yang, Zeyu Xue, Jing Feng, and Xinwang Liu. Swiftprune: Hessian-free weight pruning for large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 12868–12879, 2025.

Dong Hoon Lee and Seunghoon Hong. Learning to merge tokens via decoupled embedding for efficient vision transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2412.10569.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

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 LLM compression and acceleration. In *Proceedings of Machine Learning and Systems (MLSys)*, 2024. URL https://arxiv.org/abs/2306.00978.

Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. QServe: W4A8KV4 quantization and system co-design for efficient LLM serving. In *Proceedings of Machine Learning and Systems (MLSys)*, 2025. URL https://arxiv.org/abs/2405.04532.

Gui Ling, Ziyang Wang, Yuliang Yan, and Qingwen Liu. SlimGPT: Layer-wise structured pruning for large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2412.18110.

Dong Liu, Shu Wang, Yanxuan Yu, Haisheng Wang, and Ben Lengerich. Csv-decode: Certifiable sub-vocabulary decoding for efficient large language model inference, 2026. URL https://arxiv.org/abs/2511.21702.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. LLM-Pruner: On the structural pruning of large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023. URL https://arxiv.org/abs/2305.11627.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations (ICLR)*, 2017.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černocký, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Proceedings of the 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5528–5531. IEEE, 2011. doi: 10.1109/ICASSP.2011.5947611.

O. Pujol, P. Radeva, and J. Vitria. Discriminant ecoc: a heuristic method for application dependent design of error correcting output codes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(6):1007–1012, 2006. doi: 10.1109/TPAMI.2006.116.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Jintian Shao, Hongyi Huang, Jiayi Wu, YiMing Cheng, ZhiYu Wu, You Shan, and MingKai Zheng. VQ-Logits: Compressing the output bottleneck of large language models via vector quantized logits, 2025. URL https://arxiv.org/abs/2505.10202.

Oliver Sieberling, Denis Kuznedelev, Eldar Kurtic, and Dan Alistarh. EvoPress: Accurate dynamic model compression via evolutionary search. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2410.14649.

Yang Song, Qiyu Kang, and Wee Peng Tay. Error-correcting output codes with ensemble diversity for robust learning in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 2021.

Shengkun Tang, Oliver Sieberling, Eldar Kurtic, Zhiqiang Shen, and Dan Alistarh. DarwinLM: Evolutionary structured pruning of large language models, 2025. URL https://arxiv.org/abs/2502.07780.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models, 2023. URL https://arxiv.org/abs/2302.13971.

Asahi Ushio, Yi Zhou, and Jose Camacho-Collados. Efficient multilingual language model compression through vocabulary trimming. In *Findings of the Association for Computational Linguistics: EMNLP 2023*. Association for Computational Linguistics, 2023. URL https://aclanthology.org/2023.findings-emnlp.981/.

Gunjan Verma and Ananthram Swami. Error correcting output codes improve probability estimation and adversarial robustness of deep neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32. Curran Associates, Inc., 2019. URL https://papers.nips.cc/paper_files/paper/2019/hash/cd61a580392a70389e27b0bc2b439f49-Paper.pdf.

Zhongwei Wan, Xin Wang, Che Liu, Samiul Alam, Yu Zheng, Jiachen Liu, Zhongnan Qu, Shen Yan, Yi Zhu, Quanlu Zhang, Mosharaf Chowdhury, and Mi Zhang. Efficient large language models: A survey. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=bsCCJHbO8A.

Li-Na Wang, Hongxu Wei, Yuchen Zheng, Junyu Dong, and Guoqiang Zhong. Deep error-correcting output codes. *Algorithms*, 16(12), 2023. ISSN 1999-4893. doi: 10.3390/a16120555. URL https://www.mdpi.com/1999-4893/16/12/555.

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 *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023. URL https://arxiv.org/abs/2211.10438.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report, 2024. URL https://arxiv.org/abs/2407.10671.

Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, Xiaoxia Wu, Conglong Li, and Yuxiong He. ZeroQuant: Efficient and affordable post-training quantization for large-scale transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. URL https://arxiv.org/abs/2206.01861.

Anlan Yu, Ning Lyu, Jieming Yin, Zhiyuan Yan, and Wujie Wen. Cola: Orchestrating error coding and learning for robust neural network inference against hardware defects. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 40277–40289. PMLR, 2023. URL https://proceedings.mlr.press/v202/yu23a.html.

Hanling Zhang, Yayu Zhou, Tongcheng Fang, Zhihang Yuan, Guohao Dai, Wanli Ouyang, and Yu Wang. Vocabtailor: Dynamic vocabulary selection for downstream tasks in small language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. URL https://aclanthology.org/2026.findings-acl.1418/.

Mingyang Zhang, Hao Chen, Chunhua Shen, Zhen Yang, Linlin Ou, Xinyi Yu, and Bohan Zhuang. LoRAPrune: Structured pruning meets low-rank parameter-efficient fine-tuning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3013–3026, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.178. URL https://aclanthology.org/2024.findings-acl.178/.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. Opt: Open pre-trained transformer language models, 2022.

Weilin Zhao, Tengyu Pan, Xu Han, Yudi Zhang, Ao Sun, Yuxiang Huang, Kaihuo Zhang, Weilun Zhao, Yuxuan Li, Jie Zhou, Hao Zhou, Jianyong Wang, Zhiyuan Liu, and Maosong Sun. FR-spec: Accelerating large-vocabulary language models via frequency-ranked speculative sampling. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3909–3921, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.198. URL https://aclanthology.org/2025.acl-long.198/.

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 (MLSys)*, 2024. URL https://arxiv.org/abs/2310.19102.

Cemre Zor, Berrin Yanikoglu, Terry Windeatt, and Ethem Alpaydin. Flip-ecoc: A greedy optimization of the ecoc matrix. In *Computer and Information Sciences*, pp. 149–154, Dordrecht, 2010. Springer Netherlands. ISBN 978-90-481-9794-1.

Cemre Zor, Berrin Yanikoglu, Erinc Merdivan, Terry Windeatt, Josef Kittler, and Ethem Alpaydin. Beamecoc: A local search for the optimization of the ecoc matrix. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 198–203, 2016.

Miguel Ángel Bautista, Sergio Escalera, Xavier Baró, Petia Radeva, Jordi Vitriá, and Oriol Pujol. Minimal design of error-correcting output codes. *Pattern Recognition Letters*, 33(6):693–702, 2012. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2011.09.023. URL https://www.sciencedirect.com/science/article/pii/S0167865511002960.

A Experimental Setup

A.1 Models and Datasets

OPT-125M. We use OPT-125M (Zhang et al., 2022) ($d = 768$, 12 layers, 125M parameters) as the smallest architecture in the study.

Qwen2-0.5B. We use Qwen2-0.5B (Yang et al., 2024) ($d = 896$, 24 layers), which uses grouped-query attention and SwiGLU feed-forward blocks.

Mistral-7B. We use Mistral-7B v0.1 (Jiang et al., 2023) ($d = 4096$, 32 layers), with grouped-query attention and sliding-window attention, as the largest architecture in the TinyStories experiments.

TinyStories. TinyStories (Eldan & Li, 2023) contains approximately 2.1 million training stories and 22K validation stories. For each architecture, we evaluate vocabularies of 1,000, 5,000, and the full model vocabulary, formed from the most frequent training tokens. The same vocabulary construction is used for ECOC and Softmax within each setting.

C4. For the open-domain full-vocabulary experiments we train Qwen2-0.5B from scratch on the English (`en`) configuration of C4 (Raffel et al., 2020) at the full effective vocabulary ($V = 114{,}179$). We take a single in-order pass over the C4 train shards up to a budget of about 650M tokens, so no document is seen twice, and evaluate on the disjoint official C4 validation split.

WikiText-103. We also train OPT-125M from scratch on WikiText-103 (Merity et al., 2017) at its full effective vocabulary ($V = 46{,}888$). C4 and WikiText-103 are the two open-domain corpora used for the full-vocabulary accuracy and efficiency measurements; their training settings are given in Table 6.

A.2 TinyStories Training Hyperparameters

Table 5 gives the architecture-specific settings used for the TinyStories experiments. The ECOC-specific settings are shared across architectures unless stated otherwise.

Table 5: Pre-training hyperparameters. All models use AdamW optimizer with cosine learning rate schedule, gradient clipping at 1.0, warmup ratio 0.1, weight decay 0.01, and random seed 42.

Parameter	OPT-125M	Qwen2-0.5B	Mistral-7B
Hidden dimension d	768	896	4096
Total parameters	125M	500M	7B
Batch size (per device)	64	32	64
Number of GPUs	1	1	8
GPU type	NVIDIA A10G (24 GB)	NVIDIA A10G (24 GB)	NVIDIA A10G (24 GB)
Effective batch size	64	32	512
Epochs	2	2	1
Learning rate	3×10^{-4}	3×10^{-4}	4×10^{-4}
Sequence length	256	256	128
Mixed precision	BF16	BF16	BF16
Distributed strategy	N/A	N/A	DeepSpeed ZeRO-3
Approx. training time (ECOC)	$\sim$10h	$\sim$20h	$\sim$43h
ECOC-specific			
Loss function	Cross-entropy (L1, sigmoid)		
Codebook type	Minimal + random bits (uniform $\{0, 1\}$)		
Distance temperature τ	$L/100$		
ECOC decision layer initialization	$\mathcal{N}(0, 1/d)$		
Codebook seed	42		

A.3 Open-Domain Training and Efficiency Measurement

Training and evaluation. The C4 and WikiText-103 experiments use Qwen2-0.5B and OPT-125M, respectively. All models are trained from scratch in bf16 on a single GPU with gradient checkpointing and sequence packing. Accuracy is measured on held-out validation data over $N = 150{,}000$ scored positions. The full training settings are given in Table 6; the TinyStories settings are in Appendix A.2.

Table 6: Training hyperparameters for the open-domain (C4 and WikiText-103) experiments. All arms use AdamW with a cosine schedule, 500 warmup steps, weight decay 0.01, gradient clipping at 1.0, bf16 with gradient checkpointing, and seed 42, trained from scratch on a single GPU.

Parameter	**C4 (Qwen2-0.5B)**	**WikiText-103 (OPT-125M)**
Vocabulary size V	114,179	46,888
Hidden dimension d	896	768
Training tokens	650M (English subset)	full corpus
Epochs	1	10
Effective batch size	64 (micro-batch 32×2)	128
Sequence length	256	256
Learning rate	3×10^{-4}	3×10^{-4}
Warmup steps	500	500
Weight decay	0.01	0.01
Gradient clipping	1.0	1.0
Mixed precision	BF16	BF16
Evaluation positions N	150,000	150,000
ECOC-specific		
Loss function	Cross-entropy (L1, sigmoid)	
Codebook type	Minimal + random bits (uniform $\{0, 1\}$)	
Distance temperature τ	$L/100$	
ECOC decision layer initialization	$\mathcal{N}(0, 1/d)$	
Codebook seed	42	

Baselines. Tied Softmax reuses the input embedding as the output projection. Adaptive Softmax uses a frequent-token decision layer containing 90% of cumulative token mass, with a dimensionality-reduction factor of 4 for tail clusters. Hierarchical Softmax (Goodman, 2001) uses a two-level class factorization; our implementation partitions contiguous vocabulary IDs into equal-capacity blocks and, unless overridden, uses $\lceil \sqrt{V} \rceil$ groups. Adaptive and Hierarchical Softmax therefore reduce training computation by evaluating a target-dependent branch, while retaining a vocabulary-sized input embedding.

What the efficiency quantities mean. A *trainable parameter* is updated by gradient descent. AdamW stores two additional optimizer values for each trainable parameter. The ECOC codebook is fixed: it must be stored and used whenever full-vocabulary scores are produced, but it receives no gradients and has no AdamW state. This is why the reduction in trainable parameters can be much larger than the reduction in raw memory.

We use *decision layer* for the mapping from the final hidden state to token scores. For Softmax this is the $d \times V$ projection. For ECOC it is the $d \times L$ projection followed by fixed-codebook decoding. Parameter counts and FLOPs follow from the architecture, while optimizer-state counts follow from the number of trainable parameters. Interface storage additionally depends on the stored dtypes. Peak memory and latency are measured and also depend on the execution configuration.

Training peak memory includes model weights, gradients, AdamW state, activations, logits, and temporary workspaces. Inference *interface storage* is narrower: it contains only the input embedding, output head, and fixed codebook, not the transformer backbone or runtime activations.

Principal measurement setting. Tables 3 and 4 use bf16 parameters on one NVIDIA L4 with 22.03 GiB usable under torch 2.10 and CUDA 12.8. Softmax and ECOC use dense cross-entropy that materializes the full $[N, V]$ logit tensor; Adaptive and Hierarchical Softmax use their native target-dependent losses. All training arms use gradient checkpointing, AdamW with clipping at 1.0, and the effective batch size of the corresponding accuracy run.

Both the training and inference measurements use the folded ECOC decode described in Section C.4: the affine distance-correction terms and temperature are absorbed into static decode weights and bias, avoiding full-$[N, V]$ pointwise correction passes after the codebook multiplication. For inference this is argmax-equivalent; for the training loss it is loss-equivalent, since the only dropped term is constant across the vocabulary for each row and cross-entropy is invariant to such per-row shifts, so the loss and its gradients are unchanged. Timings are CUDA-event medians over 100 steps after 20 warmup steps. Parameter and optimizer-state counts are read from the `model.safetensors` and `optimizer.pt` files saved by the accuracy runs. All arms are measured under the same rule: eager PyTorch with no custom CUDA or Triton kernels and no CUDA-graph capture for any

Table 7: Detailed inference-efficiency accounting at $L = 200$: interface storage, output FLOPs per token, and decode-layer latency across batch sizes. Adaptive-softmax FLOPs are frequency-dependent.

	computed		*measured latency (ms)*			
Output layer	MB	MFLOP/tok	$B=1$	$B=32$	$B=128$	$B=512$
C4, Qwen2-0.5B, $V=114{,}179$						
untied Softmax	409.2	204.6	0.921	1.033	1.167	3.428
tied Softmax	204.6	204.6	0.874	0.995	1.128	3.516
Adaptive softmax	245.5	40.9	0.463	0.464	0.748	3.391
Hierarchical softmax	409.8	205.2	0.952	1.213	1.766	6.465
ECOC-Softmax	250.7	46.0	0.165	0.260	0.441	0.958
ECOC-Tying	46.0	46.0	0.165	0.268	0.440	0.955
WikiText-103, OPT-125M, $V=46{,}888$						
untied Softmax	144.0	72.0	0.338	0.372	0.399	0.603
tied Softmax	72.0	72.0	0.339	0.373	0.398	0.603
Adaptive softmax	93.2	21.2	0.437	0.430	0.431	1.064
Hierarchical softmax	144.4	72.4	0.412	0.508	0.627	1.831
ECOC-Softmax	91.1	19.1	0.134	0.128	0.145	0.333
ECOC-Tying	19.1	19.1	0.127	0.122	0.143	0.329

decision layer, with per-call waste (host synchronizations, redundant allocations, and view-versus-copy inefficiencies) removed from every arm. ECOC's decode is a single dense codebook matmul with a bias epilogue and no data-dependent control flow, so it has no such waste to remove; Adaptive and Hierarchical Softmax are held to the same standard rather than being penalized for framework overhead that a fused kernel would eliminate. Sections C.2–C.4 vary the execution setting and examine implementation sensitivity.

A.4 Additional Inference Accounting

Table 3 in the main text reports the complete training accounting. The main inference table keeps only the endpoint batch sizes for readability; Table 7 restores the intermediate decode batch sizes.

Across $B \in \{1, 32, 128, 512\}$, both ECOC variants are faster than all reported baselines on both datasets. Section C.4 shows how the folded distance formulation produces this behavior and compares it with the naive score decomposition.

B Efficiency: FLOP Accounting

B.1 Detailed Forward FLOPs

The main text uses the dominant matrix-multiplication terms. Here we give the scalar-inclusive forward estimates. Let d be the hidden dimension, V the vocabulary size, L the ECOC code length, S the sequence length, and N_L the number of transformer layers.

Backbone. The transformer backbone is identical across output-layer variants. For a standard layer with GQA and a SwiGLU feed-forward network,

$$\mathrm{FLOPs}_{\mathrm{backbone}} = N_L \left(\frac{41}{2} S d^2 + 4 S^2 d \right). \tag{4}$$

Standard Softmax. The output layer uses $2dV$ FLOPs for the $d \to V$ projection, approximately $3V$ for Softmax normalization, and 2 for the cross-entropy term:

$$\mathrm{FLOPs}_{\mathrm{Softmax}} = S(2dV + 3V + 2). \tag{5}$$

ECOC-Softmax with sigmoid and L1 distance. The $d \to L$ projection costs $2dL$ FLOPs and the sigmoid approximately $4L$. For binary codewords,

$$\|\tilde{u} - m_i\|_1 = \sum_j \tilde{u}_j + \|m_i\|_1 - 2\tilde{u}^\top m_i, \tag{6}$$

so the vocabulary comparison uses a $2LV$ matrix product, an L-term row sum, and approximately $3V$ combination operations. Adding temperature scaling, normalization, and cross-entropy gives

$$\text{FLOPs}_{\text{ECOC}-\text{L1}-\text{sigmoid}} = S(2dL + 2LV + 5L + 8V + 2). \tag{7}$$

Thus the dominant comparison is $2dV$ for Softmax versus $2dL + 2LV$ for ECOC.

Other ECOC variants. With L2 distance and sigmoid,

$$\text{FLOPs}_{\text{ECOC}-\text{L2}-\text{sigmoid}} = S(2dL + 2LV + 6L + 8V + 2). \tag{8}$$

With raw logits and L2 distance,

$$\text{FLOPs}_{\text{ECOC}-\text{L2}-\text{raw}} = S(2dL + 2LV + 2L + 8V + 2). \tag{9}$$

L1 distance without sigmoid cannot use the same binary decomposition because it requires predictions in $[0, 1]$; unrestricted raw predictions therefore fall back to explicit element-wise $O(LV)$ computation.

Table 8: Forward FLOPs per token for different ECOC configurations.

Normalization	Distance	FLOPs per token
Sigmoid	L1	$2dL + 2LV + 5L + 8V + 2$
Sigmoid	L2	$2dL + 2LV + 6L + 8V + 2$
Raw	L2	$2dL + 2LV + 2L + 8V + 2$
Softmax baseline		$2dV + 3V + 2$

Inference FLOPs across model scales. Table 9 applies the same GEMM-dominant comparison to the full TinyStories vocabularies. These vocabularies are smaller than the C4 and WikiText-103 vocabularies, so their absolute reduction factors differ from the main efficiency experiment. The final column also shows that the output layer becomes a smaller fraction of total model FLOPs as the transformer grows.

Table 9: Inference output-layer FLOPs per generated token at the full *TinyStories* vocabularies across model scales, using the same GEMM-dominant convention as Table 4. "% of total" includes the transformer backbone evaluated at $S = 1$; reduction factors are relative to the Softmax baseline in the same block. Absolute counts differ from the main-text C4 and WikiText-103 values because the vocabularies are smaller.

Output layer	Output FLOPs	vs. Softmax	% of total
Softmax (OPT-125M, $V = 29.3$K)	45.0M	1×	23.7%
ECOC-Softmax $L = 200$ (OPT-125M)	12.2M	3.7× cheaper	7.8%
ECOC-Softmax $L = 300$ (OPT-125M)	18.2M	2.5× cheaper	11.2%
Softmax (Qwen2-0.5B, $V = 32.2$K)	57.8M	1×	12.8%
ECOC-Softmax $L = 200$ (Qwen2-0.5B)	13.5M	4.3× cheaper	3.3%
ECOC-Softmax $L = 300$ (Qwen2-0.5B)	20.1M	2.9× cheaper	4.9%
Softmax (Mistral-7B, $V = 17.5$K)	143.6M	1×	1.3%
ECOC-Softmax $L = 200$ (Mistral-7B)	8.8M	16× cheaper	0.08%
ECOC-Softmax $L = 300$ (Mistral-7B)	13.1M	11× cheaper	0.12%

B.2 Training Decision Layer FLOP Accounting

A uniform "three times forward FLOPs" rule is not valid for these heads. A learned matrix has forward, input-gradient, and weight-gradient matrix multiplications. The fixed ECOC codebook has a forward comparison and input gradient, but no codebook weight gradient. Adaptive and Hierarchical Softmax also execute only target-dependent branches.

Table 3 therefore uses one common GEMM-only training convention: two FLOPs per multiply–accumulate, including the forward, input-gradient, and learned-weight-gradient matrix multiplications

Table 10: Per-segment decomposition at the principal fixed setting, measured with CUDA events at module boundaries for the forward pass and tensor hooks for the backward pass. The table reports the input embedding, backbone, and decision-layer/loss segments, each summed over accumulated micro-batches. Together with gradient clipping and the optimizer update, which occur once per update, these five components reproduce the measured step to within 0.1% for every arm; Residual is the remaining relative discrepancy. With gradient checkpointing, backbone backward includes activation recomputation and is not directly comparable with a no-checkpointing measurement. All segment times are in milliseconds.

	Input emb.		Backbone		Decision layer + loss		
Output layer	fwd	bwd	fwd	bwd	fwd	bwd	Residual
C4, micro-batch 32 × 2							
untied Softmax	0.4	5.9	425.3	1370.7	285.8	441.8	-0.07%
tied Softmax	0.4	11.1	430.7	1413.4	289.3	443.8	$+0.03\%$
Adaptive softmax	0.5	5.9	458.9	1385.3	107.1	164.1	$+0.02\%$
Hierarchical softmax	0.4	5.8	432.4	1366.7	14.0	28.8	-0.04%
ECOC-Softmax $L = 200$	0.4	5.8	454.8	1391.9	140.7	176.3	$+0.06\%$
ECOC-Tying $L = 200$	1.6	**1.5**	458.0	1390.7	**140.4**	**176.4**	$+0.03\%$
WikiText-103, micro-batch 64 × 2							
untied Softmax	0.6	3.2	167.3	570.1	166.0	257.9	$< 0.01\%$
tied Softmax	0.6	5.1	168.5	572.9	166.4	256.8	-0.10%
Adaptive softmax	0.5	3.3	175.9	528.0	80.2	121.2	$+0.00\%$
Hierarchical softmax	0.5	3.2	162.2	514.8	9.8	14.7	$+0.03\%$
ECOC-Softmax $L = 200$	0.5	3.3	183.7	534.8	100.6	131.9	$+0.07\%$
ECOC-Tying $L = 200$	3.0	2.8	182.8	533.3	**100.7**	**131.9**	$+0.08\%$

in the decision-layer/loss. It excludes the backbone, input-embedding lookup or derivation, optimizer work, and scalar reductions.

Under this convention, untied and tied Softmax require $6dV$ decision layer FLOPs per scored token. Fixed-codebook ECOC requires

$$6dL + 4LV, \tag{10}$$

where $6dL$ covers the learned code-space projection in forward and backward, and $4LV$ covers the fixed codebook in forward and its input gradient. ECOC-Tying has the same output-layer count because the derived input embedding is outside this decision-layer-only metric.

For Hierarchical Softmax, the nominal target-path count is $6d(C + s)$, where C is the number of classes and s the target block size. The implemented batched matrix multiplication can do more work when target classes have unequal batch occupancy. For Adaptive Softmax, if a_0 is the shortlist size, K the number of tail classes, r_i the tail dimension, n_i the tail size, and π_i the fraction of targets in tail i, the expected count is

$$6d(a_0 + K) + \sum_i \pi_i \, 6(dr_i + r_i n_i). \tag{11}$$

The ranges in Table 3 use the stated 90% head mass and possible splits of the remaining targets across the two tail groups. Exact adaptive expectations and realized Hierarchical-Softmax work would require the scored-target distribution and batch class occupancy from each run.

The distinction is therefore simple: ECOC greatly reduces the learned vocabulary representation but still performs a full-vocabulary code comparison, whereas Adaptive and Hierarchical Softmax make their training cost depend on the target branch rather than on all V vocabulary items.

C Efficiency: Measurements and Sensitivity

C.1 Training-Step Decomposition

Table 10 decomposes the measured training step into its input-embedding, backbone, and decision-layer/loss segments, so the per-arm whole-step differences in Table 3 can be attributed to specific components.

On C4, the tied-Softmax to ECOC-Tying whole-step difference (Table 3) is 437.9 ms. Combining the two tables, this difference decomposes into 416.3 ms from the decision-layer/loss and 8.4 ms

Table 11: Gradient checkpointing (GC) on and off on WikiText-103, with micro-batch 64 and two accumulation steps. The effective batch is 128, matching the accuracy runs; this experiment varies the execution setting rather than retraining the models. "below tied" reports the peak-memory reduction in MB relative to tied Softmax, with GC on/off. N/A means not applicable for the Softmax reference rows.

	Step (ms)		Peak (MB)		GC saves	below tied
Output layer	GC on	GC off	GC on	GC off	(MB)	on / off
untied Softmax	1189.1	1023.9	12,390	16,029	3639	N/A
tied Softmax	1186.4	1018.9	12,102	15,742	3640	N/A
Adaptive softmax	928.6	757.7	7,100	10,739	3639	5002 / 5003
Hierarchical softmax	730.1	575.5	**2,181**	**5,599**	3418	9921 / 10143
ECOC-Softmax $L = 200$	974.2	774.0	7,544	11,180	3636	4558 / 4562
ECOC-Tying $L = 200$	970.1	775.8	7,271	10,905	3634	4831 / 4837

from the input-embedding segment (both differences of the tied and ECOC-Tying rows in Table 10), 15.0 ms from the optimizer update (Table 3), and 2.7 ms from gradient clipping, partly offset by the ECOC-Tying backbone being 4.6 ms slower in this measurement. ECOC-Softmax and ECOC-Tying have nearly identical decison layer totals, as expected from their shared output path.

The smaller C4 input-embedding time for ECOC-Tying does not repeat on WikiText-103, where the ECOC-Tying and ECOC-Softmax input-embedding totals in Table 10 are 5.8 and 3.8 ms (ECOC-Tying is larger, the opposite of the C4 ordering). We therefore do not treat input-embedding latency as a general ECOC result. The unchanged backbone also shows small timing variation across arms, so small whole-step differences between ECOC-Softmax and ECOC-Tying should not be interpreted as an architectural speed ordering.

C.2 Gradient-Checkpointing Sensitivity

Gradient checkpointing saves activation memory by recomputing selected activations during backward. ECOC instead reduces vocabulary-interface weights and their associated training storage. Table 11 tests whether the two savings persist together.

For five of the six arms, checkpointing saves 3634–3640 MB; Hierarchical Softmax saves 3418 MB, possibly because its peak occurs at a different point in the step. More importantly, ECOC's absolute difference from tied Softmax is almost unchanged when checkpointing is disabled: 4831 versus 4837 MB for ECOC-Tying and 4558 versus 4562 MB for ECOC-Softmax. Thus, in this setting, checkpointing and ECOC reduce largely different parts of training memory. This is a configuration-specific observation, not a claim of universal independence.

The ratios change even though the absolute ECOC-Tying memory difference is stable: the tied/ECOC-Tying peak-memory ratio is $1.66\times$ with checkpointing and $1.44\times$ without it, while the step-time ratio changes from $1.22\times$ to $1.31\times$.

C.3 Batch-Size Dependence of the Training Head

Table 12 isolates the dense decision layer to show how memory and runtime change with micro-batch size. It is not a whole-model trainability or throughput benchmark.

Across the repeatable points, the tied-Softmax/ECOC-Softmax head-time ratio is 2.11–$2.35\times$ on C4 and 1.76–$1.85\times$ on WikiText-103. The WikiText-103 Softmax point at micro-batch 128 reaches 94% of device memory and is not repeatable, so its $2.81\times$ ratio is not treated as a stable operating point.

The memory boundary is nevertheless informative for this isolated benchmark. On C4, Softmax runs at micro-batch 32 but not 64, whereas ECOC runs at 64 but not 128. On WikiText-103, Softmax is repeatable at 64 but not 128, whereas ECOC runs at 128 but not 256. Thus the largest successful ECOC-head micro-batch is twice the largest repeatable Softmax decision layer micro-batch on both fixtures. These limits are specific to the stated dense-bf16 L4 benchmark.

Table 12: Isolated synthetic decision layer forward-plus-backward measurements across micro-batches, using dense cross-entropy on one NVIDIA L4 with 22.03 GiB usable. Peak memory accompanies each timing to show proximity to device capacity. No accuracy result attaches to an individual point. *This Softmax configuration completed once but produced an out-of-memory error on repeat attempts on an idle device and is therefore not treated as a usable operating point. "–" denotes an unavailable measurement because the corresponding Softmax run did not complete.

	tied Softmax		ECOC-Softmax $L = 200$			Softmax
Micro-batch	ms	peak (MB)	ms	peak (MB)	Ratio	% of VRAM
C4, Qwen2-0.5B						
8	79.9	3489	37.9	1926	2.11×	15%
16	167.0	6759	75.3	3796	2.22×	29%
32	355.7	13289	151.6	7537	2.35×	58%
64	*out of memory*	–	311.0	15014	–	–
WikiText-103, OPT-125M						
16	51.3	2777	28.8	1577	1.78×	12%
32	100.1	5464	57.0	3118	1.76×	24%
64	209.5	10838	113.1	6199	1.85×	47%
128	635.4*	21579	226.2	12358	2.81×	**94%**

C.4 Sensitivity to Implementation Choices

The parameter counts and nominal FLOPs above are fixed by the architecture, but measured memory and latency also depend on the implementation.

Fused cross-entropy. The principal setting in Table 3 uses unfused dense cross-entropy. A fused cross-entropy (chunked linear-plus-CE, which never materializes the full $[N, V]$ score tensor and recomputes the projection in the backward pass) changes the whole-step and peak-memory comparisons in opposite directions, as shown in Table 13. The fused backward recomputes a projection of size $2dV$ for Softmax and only $2dL$ for ECOC, so the much larger Softmax projection pays far more recomputation; the whole-step advantage of ECOC-Tying over tied Softmax therefore *widens* under fusion (from $1.20\times$ to $2.46\times$ on C4 and from $1.22\times$ to $3.08\times$ on WikiText-103). At the same time fusion removes the $[N, V]$ tensor from every dense head, so the peak-memory advantage *narrows* (from $1.58\times$ to $1.30\times$ on C4 and from $1.66\times$ to $1.10\times$ on WikiText-103). Relative to Hierarchical Softmax, fused ECOC-Tying matches its whole-step time and undercuts its peak memory on C4, and closes most of the remaining gap on WikiText-103. Hierarchical Softmax has no $[N, V]$ tensor to remove, so it neither needs nor benefits from this fusion and is reported unfused; its low training footprint is architectural rather than an implementation choice. Adaptive Softmax's target-dependent pruning is likewise intrinsic to that method and remains part of its training-FLOP and step-time advantage. This illustrates why speed and memory ratios can move in different directions under different implementations.

Table 13: Whole-step time and peak memory under unfused dense cross-entropy versus fused (chunked linear-plus-CE) cross-entropy, on one NVIDIA L4 with gradient checkpointing at the effective batch size of the corresponding accuracy run. Fusion never materializes the $[N, V]$ score tensor and recomputes the projection in the backward pass, which favors the smaller ECOC projection on time and equalizes the dense heads on memory. Hierarchical Softmax forms no $[N, V]$ tensor and is reported unfused for reference.

	Whole step (ms)		Peak memory (MB)	
Output layer	unfused	fused	unfused	fused
C4, Qwen2-0.5B, effective batch 64				
tied Softmax	2646.8	4784.7	17,199	5,221
ECOC-Softmax $L = 200$	2228.5	2016.6	11,663	4,831
ECOC-Tying $L = 200$	2208.9	1946.7	10,855	4,017
Hierarchical softmax	1922.8	*n/a*	5,615	*n/a*
WikiText-103, OPT-125M, effective batch 128				
tied Softmax	1186.4	2741.7	12,102	2,586
ECOC-Softmax $L = 200$	974.2	877.5	7,544	2,534
ECOC-Tying $L = 200$	970.1	891.2	7,271	2,342
Hierarchical softmax	730.1	*n/a*	2,181	*n/a*

Decode formulation and batch size. The algebraic form of the distance decode materially affects latency. A naive decomposition scores $(2\,p\,M^\top - \sum_j M - \sum_j p)/\tau$ and applies the affine corrections and temperature as separate elementwise kernels over the full $[N, V]$ score tensor. At $B = 512$ on WikiText-103, that form takes 1.629 ms versus 0.603 ms for tied Softmax. In the folded formulation used for the main results, the centered matrix $2M-1$ supplies $-\sum_j p_j$, the bias supplies $-\sum_j M_{ij}$, and τ is absorbed into the static decode buffers. This removes the full-$[N, V]$ pointwise correction passes while preserving the same vocabulary scores in exact arithmetic.

With the folded decode, both ECOC variants are faster than all reported baselines at every measured batch size. At $B = 512$, ECOC-Tying takes 0.955 ms versus 3.516 ms for tied Softmax on C4 and 0.329 ms versus 0.603 ms on WikiText-103. In bf16, the folded formulation changes top-1 accuracy by at most 0.1 percentage points and perplexity by 0.2% relative to the uncentered decode (measured at $L = 200$ and full vocabulary).

FLOP counts also do not determine latency by themselves. On C4, adaptive softmax uses about 11% fewer output FLOPs than ECOC but is $2.8\times$ slower at $B = 1$, corresponding to about 88 versus 279 GFLOP/s. Its clustered, data-dependent execution and ECOC's dense matrix operations are a plausible explanation, but this benchmark does not isolate the cause.

Pruned top-k for Adaptive and Hierarchical Softmax. At single-token decoding, Adaptive and Hierarchical Softmax need not score the whole vocabulary: because a cluster's (class) log-probability upper-bounds the log-probability of every token it contains, one can expand clusters in descending order and stop once the largest unexpanded cluster cannot beat the current k-th candidate. We implemented this exact pruned top-k for adaptive and Hierarchical Softmax and verified that it reproduces the dense full-vocabulary top-k (in FP32; BF16 differs only by rounding at the k-th boundary). On real WikiText-103 hidden states, Hierarchical Softmax expands about six of its 217 clusters per token to obtain an exact top-5, moving roughly 2 rather than 72 MB (about $34\times$ on WikiText-103, and $54\times$ on C4, where the codebook is larger). The arithmetic saving is therefore real but of order 10^1, not 10^2, and it does not translate into lower decode latency in eager execution. We built the pruned decode in two structurally different forms: an incremental one-cluster-per-round expansion, which is bound by the number of rounds because each round ends in a host synchronization; and a single-dispatch variant that speculatively expands a fixed budget of top clusters, verifies the pruning bound once, and falls back exactly only for the few rows that need more. The single-dispatch form is about twice as fast, but neither comes within an order of magnitude of the byte-bound ceiling. Measured on the same real hidden states and timing loop at $B = 1$ on WikiText-103, the better (single-dispatch) form takes 0.94 ms, versus 0.46 ms for dense Hierarchical Softmax and 0.16 ms for the ECOC decode; the residual is framework dispatch overhead (kernel launches and a certificate synchronization on an otherwise near-empty single-token step), not arithmetic. Closing it would require a fused decode kernel, which no arm in this comparison receives. At large batch there is little to prune, since the union of clusters needed across a batch covers most of the tree, so dense scoring is the appropriate choice there regardless. We therefore report the dense decode for these baselines in Table 4. The ECOC decode is already a single dense codebook matmul and has no comparable overhead to remove.

Unmeasured optimizations. Two potential optimizations are not included in the measured claims. Under data parallelism, gradient all-reduce volume is expected to scale with the number of learned parameters. At inference, bit-packing the codebook and using XNOR-popcount decoding would reduce the C4 interface-storage estimate from 46.0 to 4.3 MB. Neither path is implemented here, so we report no associated latency or end-to-end throughput result.

C.5 Scope of the Efficiency Measurements

These notes qualify the efficiency tables specifically; the paper's overall limitations are discussed in the main text.

No end-to-end throughput measurement. Timings measure on-device computation for individual steps. They exclude data loading, tokenization, checkpoint I/O, and distributed communication, so the reported ratios are not end-to-end training throughput.

Table 14: Top-1 / top-5 accuracy (%) on TinyStories for ECOC-Softmax at all recorded code lengths and four Softmax-family baselines, across three architectures and three vocabulary scales. "min" is $L = \lceil \log_2 V \rceil$; entries marked "–" were not recorded. Best top-1 and top-5 values in each column are bolded; ties are bolded throughout.

	OPT-125M			**Qwen2-0.5B**			**Mistral-7B**		
Output layer / L	1K	5K	Full	1K	5K	Full	1K	5K	Full
min	49.6 / 56.0	50.8 / 57.0	50.9 / 57.2	52.8 / 59.1	54.5 / 60.7	55.0 / 61.0	48.5 / 56.1	51.7 / 58.1	52.3 / 58.6
50	60.8 / 84.2	61.1 / 79.6	61.7 / 79.3	62.4 / 87.0	64.0 / 83.3	64.2 / 82.6	61.5 / 84.8	61.7 / 79.8	61.5 / 79.0
100	61.5 / 88.6	62.4 / 84.8	62.3 / 83.8	62.7 / 90.7	64.7 / 87.6	64.8 / 86.6	61.4 / 88.4	62.5 / 84.8	62.5 / 84.2
150	62.0 / 89.6	62.7 / 86.4	63.1 / 86.0	63.4 / 91.1	64.8 / 89.0	65.0 / 88.5	60.9 / 88.5	62.9 / 86.5	63.1 / 86.2
200	62.2 / 89.8	62.8 / 87.1	63.2 / 86.8	63.6 / 91.3	65.0 / 89.6	64.9 / 89.0	61.0 / 88.8	63.4 / 87.3	63.6 / 87.2
250	62.3 / 89.9	63.3 / 87.8	63.4 / 87.3	63.7 / 91.3	65.1 / 89.8	65.0 / 89.4	61.4 / 89.0	63.5 / 87.7	63.8 / 87.7
300	62.3 / 89.9	63.3 / 87.9	63.3 / 87.5	63.8 / 91.4	65.1 / 89.9	64.9 / 89.6	61.5 / 89.2	64.6 / 88.7	63.9 / 88.0
500	62.4 / 90.0	63.5 / 88.1	63.3 / 87.7	64.0 / 91.5	65.2 / 90.1	65.0 / 89.8	61.8 / 89.4	64.1 / 88.4	64.3 / 88.5
750	**62.6 / 90.1**	63.6 / 88.2	63.4 / 87.8	64.0 / 91.6	65.3 / 90.2	65.1 / 90.0	62.3 / 89.8	64.4 / 88.6	64.4 / 88.5
1000	**62.6 / 90.1**	63.6 / 88.2	63.5 / 87.9	64.1 / 91.6	65.4 / 90.3	65.3 / 90.0	62.8 / 90.0	64.6 / 88.8	64.6 / 88.7
untied Softmax	*62.4 / **90.1***	***64.0 / 88.6***	*64.2 / 88.6*	***64.5 / 91.8***	***66.0** / 90.6*	*66.4 / 90.7*	***63.0 / 90.3***	***65.8 / 89.9***	***65.9 / 89.9***
tied Softmax	*62.4 / 90.0*	***64.0 / 88.6***	***64.8 / 89.0***	*64.2 / 91.7*	***66.0 / 90.7***	***66.6 / 90.9***	–	–	–
Adaptive Softmax	*62.4 / 90.0*	*63.8 / 88.3*	*64.2 / 88.4*	*64.2 / 91.6*	*65.9 / 90.5*	*66.2 / 90.5*	–	–	–
Hierarchical Softmax	–	–	–	*63.87 / 91.46*	*65.62 / 90.42*	*65.92 / 90.44*	–	–	–

Single training seed and within-run timing summaries. Accuracy runs use seed 42, and timings are medians over steps within a benchmark run. We do not report variation across independent training seeds or benchmark processes.

Fixed batch configuration. Whole-step efficiency is measured at a single effective batch of 128. On WikiText-103 this uses micro-batch 64 with two accumulation steps, because a single micro-batch of 128 with the common dense cross-entropy exceeds the L4 memory for the Softmax arms. Because step time and peak memory depend on the micro-batch and accumulation schedule, the reported ratios are specific to this configuration and need not hold at other batch sizes.

Synthetic batch-size experiment. Table 12 isolates the decision layer. Its memory limits and timings do not establish whole-model trainability or end-to-end throughput.

No comparative calibration evaluation. Appendix D.2 reports perplexity, but we do not evaluate probability calibration under one common per-arm calibration protocol. The cross-arm conclusions therefore concern ranking quality rather than calibration.

D Additional Accuracy Results

D.1 Full TinyStories Accuracy Across Code Lengths

Table 14 reports the complete TinyStories ECOC-Softmax sweep. The main text shows representative code lengths so that the main-table comparisons can be read directly there.

Across the three recorded settings $L \in \{200, 250, 300\}$, ECOC-Softmax is within 2 top-1 points of untied Softmax in 23 of 27 architecture–vocabulary configurations. The four larger gaps are all for Mistral-7B at the 5K or full vocabulary and range from 2.1 to 2.4 points. Beyond this range, gains are small: from $L = 300$ to $L = 1000$, the largest top-1 increase across the nine configurations is 1.3 points.

D.2 Perplexity

Table 15 reports TinyStories perplexity, defined as the exponential of the token-weighted mean negative log-likelihood. ECOC uses $\tau = L/100$ because L1 distances accumulate over L bits. Temperature changes the predicted probabilities but not token ranking, so these values provide calibration context under the stated temperature and are not used as the primary cross-arm comparison.

Table 15: TinyStories perplexity for ECOC-Softmax and Softmax. ECOC entries use $\tau = L/100$ for their respective code lengths. Perplexity is reported for calibration context, not as the primary cross-arm metric.

	OPT-125M			**Qwen2-0.5B**			**Mistral-7B**		
L	1K	5K	Full	1K	5K	Full	1K	5K	Full
min	54.16	58.28	59.29	42.72	43.59	43.59	50.41	50.32	49.67
50	7.34	11.32	12.67	6.11	8.40	9.66	7.04	11.00	12.65
100	5.06	7.50	9.51	4.33	7.44	7.36	5.13	6.92	8.80
150	4.44	6.40	7.98	3.95	6.31	6.22	4.76	5.90	7.34
200	4.22	5.89	7.23	3.79	5.70	5.89	4.49	5.31	6.52
250	4.09	5.36	6.78	3.69	4.53	5.58	4.31	5.35	6.03
300	4.04	5.21	6.54	3.64	4.39	5.40	4.21	4.78	5.79
500	3.90	4.83	6.02	3.54	4.12	5.01	4.04	4.62	5.18
750	3.83	4.64	5.76	3.49	3.99	4.77	3.88	4.43	4.97
1000	3.81	4.58	5.60	3.48	3.93	4.66	3.78	4.27	4.78
Softmax	*3.75*	*3.95*	*4.05*	*3.35*	*3.47*	*3.51*	*3.35*	*3.52*	*3.53*

Table 16: Illustrative text samples from Mistral-7B at full TinyStories vocabulary after one training epoch, using temperature 0.8, top-$k = 10$ and top-$p = 0.95$. Samples are truncated for space and are not intended as evidence of equivalent generation quality.

Prompt	**untied Softmax**	**ECOC-Softmax** ($L = 300$)
Once upon a time	...there was a little girl named Lily. She loved to draw pictures of animals and flowers. One day, she drew a picture of a big, green frog...	...there was a little girl named Lily. She loved to play with her toys and her favorite toy was a teddy bear. One day, Lily's mommy asked her to clean up her toys...
One day, a boy named	...Tim found a long rope in his yard. He wanted to play with it. Tim saw a big tree and thought it would be fun to tie the rope to the tree...	...Tim went to the park with his mom. They saw a big tree with a hole in it. Tim looked inside and saw a small, scared bird...
There was a magical	...garden with lots of beautiful plants. One day, a small girl called Daisy saw something special. She found a big, shiny crystal...	...garden. Every day, the flowers would bloom and the birds would sing. One day, a little girl named Lucy came to the garden...

D.3 Qualitative Generation Samples

Table 16 gives illustrative Mistral-7B continuations at the full TinyStories vocabulary for untied Softmax and ECOC-Softmax ($L = 300$). The examples are qualitative only and are not evidence of equivalent generation quality.

E Method Details

E.1 Distance Scaling

The distance-based loss converts negative codeword distances into token scores. With a dense binary codebook, each distance accumulates contributions from all L bits, so its scale grows with code length. Two mechanisms control this: sigmoid-bounding the bit predictions and scaling the distance temperature with L.

Connection to the signed formulation. Let $\tilde{u} = \sigma(u)$ and define the centered prediction $q = 2\tilde{u} - 1 \in [-1, 1]^L$ and signed codeword $c_i = 2m_i - 1 \in -1, 1^L$. Since $q_\ell = 2\sigma(u_\ell) - 1 = \tanh(u_\ell/2)$, q_ℓ has the same sign as u_ℓ and $q_\ell = 0$ exactly where $u_\ell = 0$. Thus, this transformation preserves the superclass-aligned latent boundaries $\mathcal{H}_\ell$ defined in Section 3. Moreover,

$$|\tilde{u}_\ell - m_{i\ell}| = \frac{1 - c_{i\ell} q_\ell}{2},$$

and therefore

$$\|\tilde{u} - m_i\|_1 = \frac{L}{2} - \frac{1}{2} c_i^\top q.$$

Thus, minimizing binary L1 distance is equivalent to maximizing the signed dot product $c_i^\top q$. Since all signed codewords satisfy $|c_i| = \sqrt{L}$ and $|q|$ is common across classes for a fixed input, this gives the same class ordering as cosine similarity between q and the signed codewords.

Distance scale under sigmoid. With $\tilde{u}_j = \sigma(u_j) \in [0, 1]$ and $m_{ij} \in 0, 1$, each L1 term satisfies $|\tilde{u}j - mij| \leq 1$, so the total distance is bounded by L. Under an independent balanced random-code assumption, each bit contributes expected distance $1/2$, giving mean total distance $L/2$ and fluctuations of order $O(\sqrt{L})$. The variance remains bounded because each term has bounded support. In contrast, raw dot-product and L2 scores are more sensitive to the magnitude and distribution of the predicted activations. This reduced scale sensitivity helps explain why one simple temperature rule works across the tested code lengths.

Temperature scaling. With fixed τ, increasing L increases the magnitude of the negative distances entering the vocabulary Softmax and makes the distribution more peaked, concentrating gradients on a smaller set of near competitors. Scaling $\tau = L/L_{\text{ref}}$ keeps this scale approximately stable. We use $L_{\text{ref}} = 100$, i.e. $\tau = L/100$, in the reported experiments. Table 17 shows the interaction between this temperature scaling and the initialization of the projection matrix (W) for $L = 1000$ and $V = 1000$.

E.2 d-NORMALIZED INITIALIZATION

For the projection from hidden states to code predictions, a fixed weight standard deviation makes the output variance grow with hidden width d. Under standard independence assumptions, $\text{std}(u_j) \propto \sqrt{d}$. This can push the sigmoid toward saturation at initialization for wider models and reduce the initial gradient magnitude.

We therefore initialize the projection with standard deviation $1/\sqrt{d}$, equivalently variance $1/d$, so its output scale is approximately independent of hidden width. Table 17 shows the effect most clearly for Mistral-7B: with fixed $\text{std} = 0.02$ and $\tau = 1$, top-1 is 49.6%, whereas OPT-125M and Qwen2-0.5B change by less than one point relative to $\tau = 10$. d-normalized initialization raises the Mistral $\tau = 1$ result to 58.6%. With $\tau = 10$, the fixed and normalized initializations reach 62.2% and 62.8%, respectively. The two mechanisms therefore address different sources of scale variation: initialization across model widths and distance accumulation across code lengths.

Table 17: Effect of distance temperature scaling and d-normalized initialization on 1,000-bit ECOC ($V = 1{,}000$). Top section: fixed init (std $= 0.02$); bottom section: d-normalized init (std $= 1/\sqrt{d}$).

			$\tau = 1$		$\tau = L/L_{\text{ref}} = 10$	
Init	**Model**	d	Top-1	Top-5	Top-1	Top-5
std $= 0.02$	OPT-125M	768	62.6%	90.2%	62.8%	90.3%
	Qwen2-0.5B	896	63.4%	91.2%	64.1%	91.6%
	Mistral-7B	4096	49.6%	79.8%	62.2%	89.7%
std $= 1/\sqrt{d}$	OPT-125M	768	61.0%	89.2%	62.6%	90.1%
	Qwen2-0.5B	896	63.2%	91.1%	64.1%	91.6%
	Mistral-7B	4096	58.6%	87.9%	62.8%	90.0%

E.3 ERROR-CORRECTION CAPACITY

Table 18 compares the predicted and empirical minimum Hamming distance for $V = 1{,}000$. Here the minimal code length is 10 bits and $r = L - 10$ is the number of additional random bits. The error-correction capacity is $\lfloor (d_{\min} - 1)/2 \rfloor$, and the predicted minimum distance uses Equation 2.

Table 18: Minimum Hamming distance and error correction capacity for $V = 1{,}000$ with varying code lengths. Predicted $d_{\min}$ uses the bound from Equation 2. Capacity $= \lfloor (d_{\min} - 1)/2 \rfloor$ correctable bit errors.

r	Total L	Pred. $d_{\min}$	Emp. $d_{\min}$	Pred. capacity	Emp. capacity
0	10	1	1	0	0
40	50	4	8	1	3
90	100	21	28	10	13
140	150	40	48	19	23
190	200	60	65	29	32
240	250	81	90	40	44
290	300	102	110	50	54
490	500	189	201	94	100
990	1000	415	427	207	213

F Ablations

F.1 Design Ablations

Codebook design. We compare random codebooks with BCH codes (Bose & Ray-Chaudhuri, 1960), which give a constructed minimum-Hamming-distance guarantee. At matched length 127 and similar correction strength, BCH(127, 21) has nominally stronger correction than the random code used here (empirical capacity approximately 18 bits), but the downstream accuracies are effectively the same. On Qwen2-0.5B with $V = 1{,}000$, 0.4M training samples, and one epoch, the random codebook gives 58.8% top-1 / 87.8% top-5, versus 58.7% / 87.3% for BCH. We therefore do not observe an accuracy advantage from the structured code in this setting.

Distance metric and normalization. The main experiments use L1 distance with sigmoid-bounded bit predictions. L2 gives nearly identical accuracy: across the Qwen2-0.5B full-vocabulary settings in Table 19, top-1 differs by at most 0.2 points. Raw logits also gave similar accuracy when paired with an adjusted temperature, consistent with the scale analysis in Appendix E.1.

L1 is retained as the default because it is simple and matches the binary code structure. As derived in Section 3, its binary-code form gives an affine token score and therefore hyperplane decision boundaries. Its subgradient with respect to each bounded bit has constant magnitude away from zero, whereas L2 gradients shrink as predictions approach the codeword. A dot-product score on a $\{0, 1\}$ codebook has gradient m_{ij} and therefore gives no gradient through positions where the compared codeword bit is zero. These gradient properties motivate the L1 default rather than establishing an accuracy advantage, since L1 and L2 perform similarly in the ablation.

Table 19: L1 vs L2 distance: top-1 accuracy (%) on Qwen2-0.5B (full vocabulary, $\tau = L/100$). Differences are within 0.2 percentage points across all bit counts; – indicates Softmax reference remains the same between L1 and L2 columns.

Bits	**L1 Top-1**	**L2 Top-1**	**L1 Top-5**	**L2 Top-5**
15 (min)	55.0	54.9	61.0	60.9
100	64.8	64.8	86.6	86.8
200	64.9	65.0	89.0	89.0
300	64.9	65.1	89.6	89.6
500	65.0	65.0	89.8	89.9
1000	65.3	65.4	90.0	90.0
Softmax	*66.4*	–	*90.7*	–

Mixed precision. One implementation detail matters once code lengths grow. The affine L1 decoder uses the codeword Hamming weight $\|\mathbf{m}_v\|_1$, an integer between 0 and L. BF16 represents all integers exactly only through 256, so caching these weights in BF16 becomes unsafe once $L > 256$. In our implementation, this is harmless at $L = 200$ but severe at $L = 500$: caching the code matrix and derived Hamming weights in BF16 reduced top-1 from 0.378 to 0.020, while training itself was unaffected. The binary code matrix $\mathbf{M}$ is represented exactly in BF16; the issue is the reduction used to obtain its row sums, not storage of the binary entries. For $L > 256$, the row sums should therefore

be accumulated in an integer or FP32 type. Our dense implementation stores the resulting length-V vector in FP32, which is about 0.2 MB at $V \approx 100$K and less than 0.5% of the output-interface footprint. A bit-packed XNOR-popcount decoder could recover the same counts directly without storing this vector separately.

F.2 Early Training Loss Across Code Lengths

Table 20 compares independent Mistral-7B runs at different code lengths. The step-50 loss reflects the chosen codebook, code length, and early optimization dynamics; it should not be interpreted as a direct measure of Hamming distance.

Table 20: Training loss after 50 optimizer updates and final accuracy for Mistral-7B at full TinyStories vocabulary ($V = 17{,}518$), evaluated across code lengths. The relationship is empirical: step-50 loss reflects early optimization as well as the chosen code length and codebook.

Bits	Step-50 Loss	Final Top-1	Final Top-5
15 (min)	96.08	52.3%	58.6%
50	67.75	61.5%	79.0%
100	66.32	62.5%	84.2%
150	61.74	63.1%	86.2%
200	61.15	63.6%	87.2%
250	60.35	63.8%	87.7%
300	60.41	63.9%	88.0%
500	58.36	64.3%	88.5%
1000	57.87	64.6%	88.7%

The early-loss curve and final accuracy show the same broad diminishing-return pattern in this configuration: most of the improvement occurs between 15 and 200 bits, while gains beyond 300 bits are smaller. This is an empirical diagnostic, not a theoretical rule for selecting L. A short pilot across candidate code lengths may help prioritize longer runs, but the relationship should be rechecked for each model, dataset, and training configuration.