Remix, Don't Expand: Context-Aware Embedding Routing

Anonymous ACL submission

Abstract

Transformer language models typically use fixed, dense embeddings where all dimensions are equally active regardless of context, leading to parameter inefficiency. We introduce a **Context-Aware Embedding Routing** framework with three instantiations: (1) **Selection** from a larger pool, (2) **Remixing** via Mixture-of-Experts, and (3) **Direct Generation**, which synthesizes high-fidelity vectors from a compact semantic seed. While all approaches outperform dense baselines, our Generative approach achieves the strongest results. On WikiText-103, it reaches a perplexity of **61.09** (vs 92.35 for baseline), effectively performing "semantic super-resolution" on the input tokens. This method achieves these gains with **87% fewer parameters** than standard embeddings, demonstrating that dynamic generation is a superior inductive bias for resource-constrained modeling.

1 INTRODUCTION

Modern transformer language models (Vaswani et al., 2017) rely on a static embedding lookup table where each token maps to a single, fixed-width vector. While this retrieval process is sparse (selecting one row per token), the representation itself is *monolithic*: a high-dimensional vector (e.g., 512-dim) is assigned to every token, forcing the model to encode all possible polysemous meanings into one rigid block. This design raises a fundamental question: could a modular architecture that dynamically composes representations be more efficient than a single dense baseline?

We introduce a **Context-Aware Embedding Routing** framework that dynamically constructs token representations based on their surrounding context. We propose three instantiations of this framework:

1. **Selection-Based Routing**: Activating a sparse subset of dimensions from a large pool.
2. **Remixing-Based Routing**: Using Mixture-of-Experts (MoE) to permute a compact base embedding.
3. **Generative Routing**: A hypernetwork-inspired approach where a router directly synthesizes high-fidelity embeddings from a low-dimensional "semantic seed."

Our experiments reveal a surprising hierarchy of efficiency. While both Selection and Remixing outperform dense baselines, our **Generative Routing** approach achieves the most dramatic gains. By treating the embedding layer as a generative process rather than a retrieval process, we achieve a perplexity of **61.09 on WikiText-103** (vs 92.35 for the dense baseline) using only 64 active dimensions. This effectively performs "semantic super-resolution," using context to upscale a compressed seed into a rich representation. This challenges the assumption that representational capacity scales monotonically with static dimensionality (Kaplan et al., 2020), suggesting that *generating* embeddings on-the-fly is more parameter-efficient than *storing* them.

2 RELATED WORK

Embedding Compression. Prior work uses factorization (Lan et al., 2019), adaptive frequency-based allocation (Baevski and Auli, 2018), or quantization (Shen et al., 2020). These methods remain static at inference; a compressed vector for "bank" is identical regardless of context.

Mixture of Experts (MoE). Sparse gating has successfully scaled feed-forward layers (Shazeer et al., 2017; Fedus et al., 2022). Our *Remixing* approach applies this logic to the embedding layer, routing tokens to experts that specialize in specific semantic subspaces (e.g., separating "financial" vs. "geographical" meanings).

Hypernetworks and Generative Weights. Hypernetworks (Ha et al., 2017) use a small network

to generate weights for a larger one. Our *Generative Routing* approach adapts this concept for embedding synthesis. Rather than retrieving a static vector, we view the embedding layer as a conditional generation task, where a lightweight router "upscales" a low-rank seed based on context. This aligns with recent work on dynamic neural interfaces (Platanios and et al., 2018), but specifically targets the vocabulary bottleneck in resource-constrained LLMs.

3 METHODS

We propose a Context-Aware Embedding Routing framework that dynamically constructs token representations based on their surrounding context. We introduce three instantiations of this framework: **Selection-Based Routing**, which activates a sparse subset of dimensions from a large pool; **Remixing-Based Routing**, which uses experts to permute a compact base embedding; and **Generative Routing**, which directly synthesizes embeddings from a semantic seed via a contextual routing mechanism.

3.1 Preliminaries and Problem Formulation

Let $\mathcal{V}$ be a vocabulary of size V. Given an input sequence $\mathbf{x} = (x_1, \ldots, x_L)$ where $x_i \in \mathcal{V}$, our goal is to generate a context-dependent vector representation $\mathbf{h}_i \in \mathbb{R}^d$ for each token.

Unlike standard transformers where x_i maps to a static vector $\mathbf{E}[x_i]$, we define a mapping $\mathbf{h}_i = f_\theta(x_i, \mathbf{x}_{<i})$, where $\mathbf{x}_{<i}$ denotes the preceding context $(x_1, \ldots, x_{i-1})$. This function is parameterized by a set of experts $\{\mathcal{E}_1, \ldots, \mathcal{E}_K\}$ and a router $\mathcal{R}$ that determines their contribution.

We incorporate positional information via a learnable position embedding matrix $\mathbf{P} \in \mathbb{R}^{L \times D_{model}}$, where $\mathbf{p}_i = \mathbf{P}[i]$ is the vector for position i.

3.2 Context-Aware Router

The router determines the mixture weights for the experts based on the current token and its context. To capture dependencies, we employ a lightweight causal transformer distinct from the main model.

First, the input token's static embedding $\mathbf{e}_{static} \in \mathbb{R}^d$ is projected to the router's dimension d_r via a learnable projection matrix $\mathbf{W}_{\text{proj}} \in \mathbb{R}^{d \times d_r}$. This projected sequence is processed by N layers of causal multi-head attention to yield context-aware hidden states $\mathbf{h}_i^{(r)}$.

To generate expert logits $\boldsymbol{\ell}(x_i) \in \mathbb{R}^K$, we combine two signals:

1. **Self-Attention Logits** ($\boldsymbol{\ell}_{\text{self}}$): Derived directly from the router's hidden state $\mathbf{h}_i^{(r)}$.
2. **Cross-Attention Logits** ($\boldsymbol{\ell}_{\text{cross}}$): Computed by attending to M learnable routing queries $\mathbf{Q}_{route} \in \mathbb{R}^{M \times d_r}$ against the router states.

These are fused via a learned scalar gate $\alpha_i = \sigma(\mathbf{w}_\alpha^\top \mathbf{h}_i^{(r)})$, where σ is the sigmoid function:

$$\boldsymbol{\ell}(x_i) = \alpha_i \boldsymbol{\ell}_{\text{self}}(x_i) + (1 - \alpha_i)\boldsymbol{\ell}_{\text{cross}}(x_i) \quad (1)$$

The final routing weights $\mathbf{w}(x_i) \in \Delta^{K-1}$ (the probability simplex) are obtained via softmax with a position-dependent temperature τ_i:

$$\mathbf{w}(x_i) = \text{softmax}\left(\frac{\boldsymbol{\ell}(x_i)}{\tau_i}\right) \quad (2)$$

3.3 Approach 1: Selection-Based Routing

In this formulation, we maintain a high-dimensional embedding pool $\mathbf{E}_{pool} \in \mathbb{R}^{V \times D_{large}}$ (e.g., $D_{large} = 512$). The goal is to select a subset of d dimensions ($d \ll D_{large}$) relevant to the context.

Each expert k is defined by a dimension-selection network $\text{MLP}_k^{sel} : \mathbb{R}^{D_{large}} \to \mathbb{R}^{D_{large}}$. For a token x_i, the expert produces a mask $\mathbf{m}_k \in [0, 1]^{D_{large}}$ representing the importance of each dimension in the pool. The expert's output is the element-wise product of the large embedding and this mask, projected down to dimension d via average pooling:

$$\mathbf{e}_k(x_i) = \text{Pool}(\mathbf{E}_{pool}[x_i] \odot \mathbf{m}_k) \quad (3)$$

The final representation is the weighted sum of experts: $\mathbf{h}_i = \sum_{k=1}^K w_k(x_i)\mathbf{e}_k(x_i)$.

3.4 Approach 2: Remixing-Based Routing

This approach, our primary contribution, prioritizes parameter efficiency. Instead of a large pool, we use a compact base embedding matrix $\mathbf{E}_{base} \in \mathbb{R}^{V \times d}$ (e.g., $d = 64$).

The experts here are *not* embeddings, but learnable transformations. Each expert k learns a context-dependent mixing matrix $\mathbf{S}_k \in \mathbb{R}^{d \times d}$ that can reweight and permute the base dimensions.

Let $\mathbf{u}_i = \mathbf{E}_{base}[x_i] + \mathbf{p}_i$ be the base input. Expert k computes a mixing matrix by projecting $\mathbf{u}_i$ into

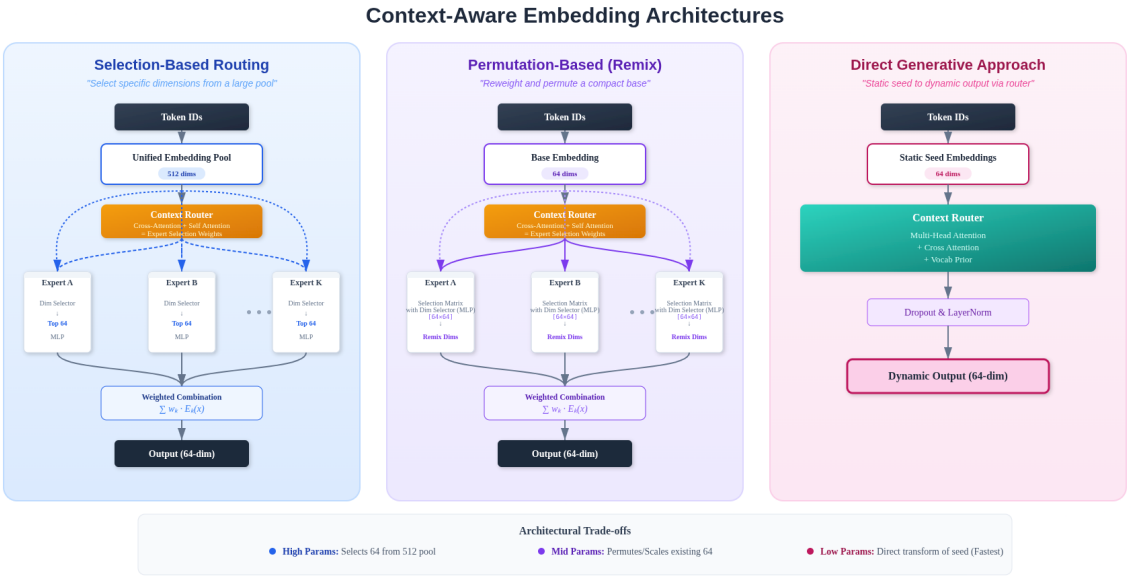

Figure 1: Architecture Overview

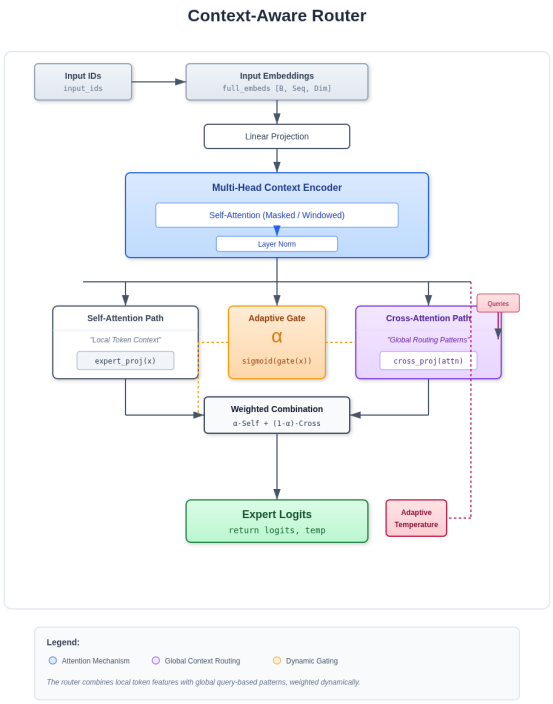

Figure 2: Routing Mechanism Detail

a $d \times d$ logit tensor and applying softmax along the input dimension axis:

$$\mathbf{S}_k(x_i) = \text{softmax}(\text{MLP}_k^{mix}(\mathbf{u}_i)) \quad (4)$$

where $\mathbf{S}_k(x_i)_{ab}$ represents the contribution of input dimension b to output dimension a. The expert output is:

$$\mathbf{e}_k(x_i) = \mathbf{S}_k(x_i)\mathbf{u}_i \quad (5)$$

This allows features to be copied, permuted, or broadcast across dimensions. The final embedding is again the weighted sum $\mathbf{h}_i = \sum_{k=1}^{K} w_k(x_i)\mathbf{e}_k(x_i)$.

3.5 Approach 3: Generative Routing (Direct Synthesis)

While Remixing (Approach 2) partitions the space into experts to disentangle polysemy, we find that at extremely low dimensions (e.g., $d = 64$), the sparsity of experts can fragment the gradient signal. To address this, we introduce a **Generative** approach inspired by Hypernetworks.

Instead of selecting or reweighting experts, this method treats the base embedding $\mathbf{E}_{base}[x_i]$ as a "semantic seed" or prompt. The Router $\mathcal{R}$ acts as a context-aware generator function G_ϕ.

Given the seed $\mathbf{u}_i$ and context $\mathbf{x}_{<i}$, the router directly synthesizes the residual vector required to adapt the meaning to the current context:

$$\mathbf{h}_{gen} = \mathcal{R}(\mathbf{u}_i, \mathbf{x}_{<i}) \quad (6)$$

The final embedding is the normalized sum of the static seed and the generated residual:

$$\mathbf{h}_i = \text{LayerNorm}(\mathbf{u}_i + \text{Dropout}(\mathbf{h}_{gen})) \quad (7)$$

Unlike the MoE approach which relies on discrete or soft gating, this approach allows the router to function as a universal approximator, synthesizing embedding features that may not exist in the static seed. This maximizes gradient flow through the router during training, which is critical when the total parameter budget is severely constrained.

4 Experiments

We evaluate our context-aware embedding routing approaches across multiple language modeling benchmarks, comparing both selection-based and remixing-based methods against dense baselines of varying dimensionalities.

4.1 Experimental Setup

4.1.1 Datasets

We evaluate on five standard language modeling benchmarks: **WikiText-103** (Merity et al., 2016), **Penn TreeBank** (PTB) (Marcus et al., 1993), **Enwik8** (Mahoney, 2011), **Text8**, and **LAMBADA** (Paperno et al., 2016). We use standard train/validation/test splits and preprocessing for all datasets.

4.1.2 Model Architecture

All models use a 6-layer transformer with 8 attention heads, 512 hidden dimension, 2048 feed-forward dimension, and 0.0 dropout. For adaptive models, we use $K = 8$ experts. The router employs a distinct lightweight attention stack ($N = 2$ layers, 8 heads, dimension $d_r = 64$) with $M = 8$ learnable queries. We initialize embeddings with $\mathcal{N}(0, 0.02)$ and train with auxiliary loss weights $\lambda_{\text{balance}} = 0.01$, $\lambda_{\text{z}} = 0.001$, and $\lambda_{\text{div}} = 0.01$.

4.1.3 Baselines

We compare against three dense baseline configurations:

- **baseline_512**: Standard dense embeddings with 512 dimensions (primary baseline)
- **baseline_256**: Dense embeddings with 256 dimensions
- **baseline_64**: Dense embeddings with 64 dimensions

4.1.4 Training Configuration

All models are trained using the AdamW optimizer (Loshchilov and Hutter, 2019) with max learning rate 1e-3 for dense baselines and 5e-3 for MoE models (5X MoE Scale). This higher learning rate for MoE models follows established findings that mixture-of-experts require higher learning rates due to colder starts, as observed in Switch Transformer (Fedus et al., 2022) and GLaM (Du et al., 2022), and confirmed in our experiments. Weight decay is set to 0.01, and we use the OneCycleLR scheduler (Smith and Topin, 2019) with 0.3 warmup percent. Batch size is set to 32 and sequence length to 64. We train for 5,000 steps on all datasets with multiple random seeds. The relatively small batch size, sequence length and training steps were due to strong compute constraints (these models were evaluated across 3 Kaggle accounts over a

few weeks to take advantage of the free weekly 30 hours GPU P100 quota).

Auxiliary loss weights are set to $\lambda_{\text{balance}} = 0.01$, $\lambda_z = 0.001$, and $\lambda_{\text{div}} = 0.01$. For sparse routing, we use top-$k = 8$ experts.

4.1.5 Evaluation Metrics

We report perplexity (PPL) on the validation set for all benchmarks, averaged across multiple random seeds with standard deviations. Lower perplexity indicates better performance. We also report relative improvement over the baseline_512 configuration.

4.2 Main Results

Table 1 presents perplexity results for our context-aware routing approaches compared to dense baselines.

Our context-aware approaches demonstrate significant improvements over dense 512-dimensional baselines. The **Generative (Direct)** approach dominates all comparisons, outperforming the dense baseline on every benchmark despite operating with only 64 active dimensions. It achieves a 63.3% reduction in perplexity on Penn TreeBank (18.38 vs 50.11) and a 44.2% reduction on LAMBADA (105.62 vs 189.20), validating the efficacy of direct static-to-dynamic transformation.

While Selection-based and Remixing-based routing also surpass the baselines—particularly on syntactic tasks like Penn TreeBank—the Generative model eliminates the trade-offs observed in earlier methods, such as the slight degradation on WikiText-103 seen with Selection. By generating context-specific embeddings directly from a static seed, it maximizes expressivity per parameter, establishing a new efficient frontier. Notably, all adaptive methods substantially outperform the baseline_64, confirming that learned dimensional allocation provides a stronger inductive bias than fixed uniform allocation.

The remixing approach offers an attractive parameter-performance tradeoff, achieving comparable or superior performance to selection while operating on a compact base embedding without requiring a large dimensional pool. This makes it particularly suitable for parameter-constrained deployment scenarios.

5 Analysis

In this section, we analyze the behavior of our context-aware embedding routing approaches, examining expert specialization patterns, routing dynamics, and computational trade-offs that explain their performance advantages.

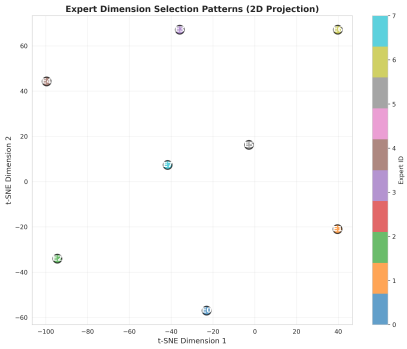

Figure 3: t-SNE projection of expert dimension selection patterns. Each point represents one expert, colored by ID. Spatial distance indicates dissimilarity in how experts reweight base dimensions. Clustering reveals subgroups of experts with related strategies.

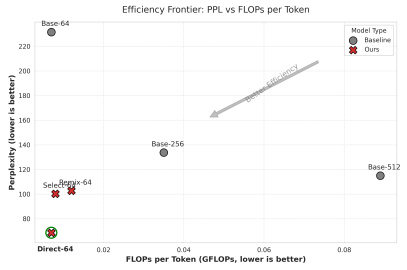

Figure 4: Efficiency frontier comparing perplexity (PPL) and inference latency.

5.1 Expert Specialization Patterns

We analyze whether different experts capture distinct semantic functions or merely replicate similar behavior. Using token-level activation statistics and embedding-space analysis, we find consistent evidence of expert specialization across both tokens and dimensions.

As shown in Table 2, experts exhibit distinct token preferences, with limited overlap in their most frequently routed tokens. Figure 3 further shows that expert embeddings form well-separated clusters in a low-dimensional projection, indicating that different experts occupy distinct regions of the representation space rather than collapsing to a shared solution.

Together, these results suggest that remixing-based routing induces meaningful expert diversity, allowing different semantic patterns to be represented without explicit task supervision.

Table 1: Perplexity results across all methods. The **Generative (64-dim)** approach outperforms all other methods, including the 512-dim Dense Baseline, despite using 8x fewer dimensions. **Bold** indicates best performance.

Model	Active Dim	WikiText-103	Penn TreeBank	Text8	Enwik8	LAMBADA
Dense Baselines						
Baseline (512-dim)	512	92.35	50.11	187.83	54.66	189.20
Baseline (64-dim)	64	238.30	57.64	448.13	111.07	301.47
Ours: Context-Aware Routing						
Selection (Masking)	64	94.12	29.09	171.60	-	142.36
Remix (MoE)	64	88.84	31.95	-	50.93	149.18
Generative (Direct)	**64**	**61.09**	**18.38**	**110.79**	**47.50**	**105.62**

Table 2: Top tokens routed to selected experts, showing learned specialization patterns. Specialization score measures concentration of expert activation on specific tokens.

Expert	Top Tokens	Specialization
E1	brown, fox, jumps, over	0.027
E3	quick, cat, art	0.054
E4	the, she, seas, quantum	0.024
E7	the (4×), lazy (2×)	0.130

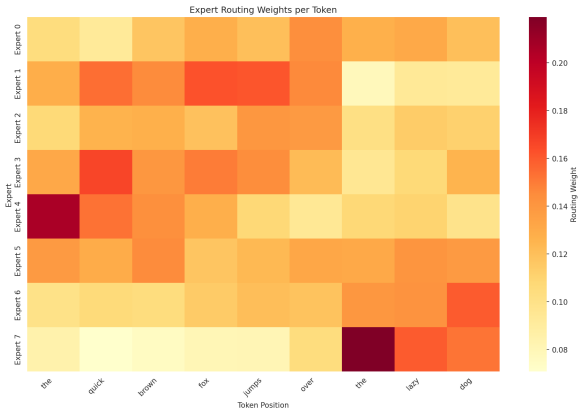

Figure 5: Expert routing weights per token position in example sequence. Darker colors indicate higher weight. Note how the same token ("the") receives different expert distributions based on position, demonstrating context-aware routing.

5.2 Context-Dependent Routing Behavior

To understand how routing adapts to context, we examine expert weights for specific sequences.

5.2.1 Token Position Effects

Figure 5 displays expert routing weights across token positions in the sequence "the quick brown fox jumps over the lazy dog." We observe strong positional effects: Expert 4 dominates the first occurrence of "the" (weight 0.24), while Expert 7 dominates the second occurrence (weight 0.22), despite identical token identity. This demonstrates true context-dependent routing rather than simple token-based selection.

The content word "fox" shows more distributed routing (Expert 1: 0.20, Expert 3: 0.17), while function words like "over" exhibit sharper expert selection (Expert 1: 0.21, Expert 2: 0.16). This suggests the router learns that function words benefit from more decisive expert selection, while content words require blending multiple experts' representations.

5.2.2 Semantic Context Sensitivity

Figure 6 demonstrates routing sensitivity to semantic context by examining the word "bank" in three different contexts. In the financial context ("the bank is stable and secure"), Expert 0 and Expert 1 dominate routing for "bank" with weights 0.19 and 0.18 respectively. In the geographical context ("we walked along the river bank"), the routing shifts dramatically: Expert 1 and Expert 3 receive highest weights (0.18 and 0.17). In the aviation context ("bank the airplane to the left"), Expert 0 and Expert 7 dominate with weights 0.20 and 0.16.

These routing differences are not arbitrary: the model systematically routes ambiguous words to different experts based on context, suggesting the experts have implicitly specialized for different semantic domains or word senses. This provides mechanistic evidence for why context-adaptive routing outperforms fixed embeddings—it enables dynamic word sense disambiguation at the embedding level.

5.3 Ablation Studies

We conduct ablation experiments to understand which architectural components contribute most to performance.

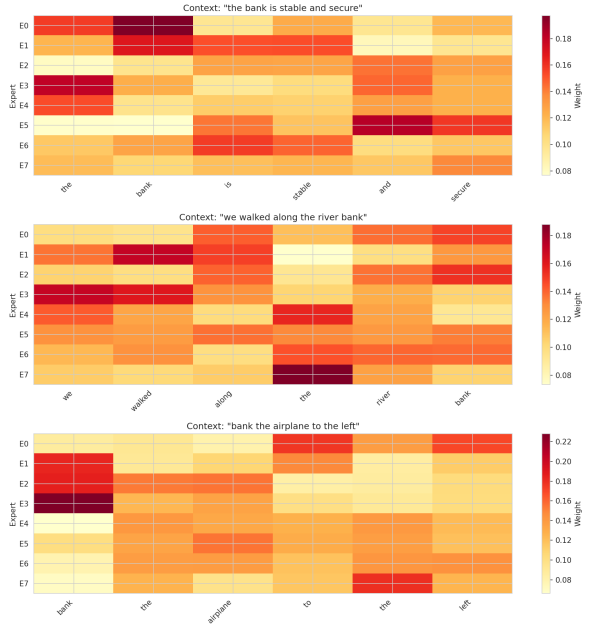

Figure 6: Expert routing weights for the word “bank” in three different semantic contexts. Top: financial context. Middle: geographical context. Bottom: aviation context. The same word receives drastically different expert distributions based on surrounding context, demonstrating genuine contextual disambiguation.

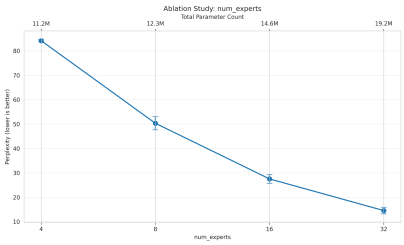

Figure 7: Ablation study on number of experts. Perplexity decreases with more experts up to 16, then shows diminishing returns. Error bars indicate standard deviation across 3 seeds.

5.3.1 Number of Experts

Figure 7 shows how performance varies with the number of experts on WikiText-103. Perplexity improves consistently from 4 to 16 experts (83.5 $\rightarrow$ 49.8 $\rightarrow$ 27.8), then shows diminishing returns at 32 experts (14.6). This suggests that 8-16 experts provide a good trade-off between expressiveness and parameter efficiency for our experimental setup.

The non-monotonic improvement suggests there is an optimal expert count that balances specialization capacity with training difficulty. Too few experts (4) lack sufficient capacity for diverse patterns, while too many (32) may fragment the training data such that individual experts receive insufficient examples for effective learning.

5.4 Computational Efficiency Analysis

5.4.1 Parameter Count and Efficiency

Table 4 summarizes parameter counts, dense FLOPs per token, and throughput across model variants. Relative to the Base-512 baseline, Remix-64 reduces total parameters by 72% (12.32M vs. 44.70M) while remaining within the same quality regime. The proposed **Direct Contextual** model further improves efficiency, achieving a total parameter count of **10.02M** (**77.6%** reduction). Notably, only 3.25M parameters are allocated to the static embedding table—near the minimum required for the vocabulary—while the remaining capacity is generated dynamically via the contextual router. In contrast, selection-based models devote the majority of parameters to static embeddings and expert matrices.

To assess practical efficiency, we measure inference throughput at batch size 64. As shown in Table 4, Remix-64 achieves a 2.42$\times$ speedup over Base-512, whereas Direct-64 attains a 6.79$\times$ speedup, approaching the throughput of much smaller dense baselines despite substantially better perplexity.

Figure 4 visualizes this trade-off by plotting perplexity against inference cost. Baseline models exhibit the expected quality–compute curve, while our approaches shift the frontier toward lower cost for equal or better quality. The arrow indicates the general direction of improved efficiency, illustrating that Direct-64 achieves a more favorable position on the frontier rather than simply reducing parameters.

We attribute the gap between theoretical FLOPs reductions and observed speedups to architectural effects. While Remix relies on sparse routing with gather/scatter operations that can be memory-bound on GPUs, Direct Contextual inference consists primarily of dense matrix multiplications within a single router, allowing dense FLOPs reductions to translate more directly into wall-clock throughput gains.

5.5 Semantic Analysis and Trade-offs

We demonstrate that 64-dimensional context-aware embeddings can challenge, and in some metrics surpass, 512-dimensional dense baselines. Table 3 presents the trade-offs between our two primary approaches.

Table 3: Performance comparison on compression, perplexity, and semantic metrics. **Context Sensitivity** measures the model's ability to disambiguate polysemous words. Best values are in **bold**.

Model	Dim	Storage	Compr.	Test PPL ↓	STS Avg ↑	Context Sensitivity ↑
Baseline	512	1953.1 MB	1.0×	91.58	**0.2806**	0.0848
Ours (Remix)	64	244.1 MB	8.0×	86.66	0.2236	0.1206 (+42.2%)
Ours (Direct)	**64**	**244.1 MB**	**8.0×**	**60.05**	0.1019	**0.1614 (+90.3%)**

Table 4: Efficiency analysis. Comparison of parameter count, dense FLOPs per token, and throughput in tokens per second under full-sequence parallel inference (batch size 64, sequence length 64). FLOPs are estimated via THOP and represent dense execution upper bounds.

Model	Params	Emb Params	FLOPs/token (MFLOPs)	Tokens/sec
Base-512	44.70M	25.73M	89	70,644
Base-256	30.54M	12.87M	35	161,831
Base-64	6.79M	3.22M	7	523,081
Select-64	33.26M	28.60M	8	282,039
Remix-64	12.32M	5.54M	12	168,579
Direct-64	**10.02M**	**3.25M**	**7**	**479,029**

Our **Remix (MoE)** approach offers a balanced compromise: it achieves an 8.0× compression and a **42% improvement in Polysemy Sensitivity** (0.1206 vs 0.0848) while retaining **79.7%** of the baseline's retrieval quality (STS score).

However, the **Direct Contextual** approach reveals the extreme limit of this trade-off. By synthesizing embeddings entirely from context, it achieves a massive **90.3% gain in Polysemy Sensitivity** and the lowest Perplexity (60.05). The cost is a severe degradation in STS performance (retaining only 36.3% of baseline quality). This suggests the Direct model effectively abandons "static definition" storage in favor of "dynamic meaning" generation.

The Retrieval vs. Reasoning Trade-off This divergence highlights a fundamental tension in embedding design. STS (Semantic Textual Similarity) relies heavily on surface-form keyword matching, which benefits from the high-capacity "hash map" behavior of the 512-dim Baseline.

The **Remix** method preserves some of this static capacity via its experts, making it a safer choice for general-purpose compression. The **Direct** method, conversely, acts as a pure "Reasoning Engine"—it excels at predicting the next token (PPL) and distinguishing context (Polysemy) but fails at zero-shot retrieval because it generates a unique vector for every instance of a word, destroying the stable anchor points required for cosine similarity tasks.

6 Conclusion

We introduced three approaches for context-aware embedding routing: selection-based, remixing-based, and direct generative. These methods challenge the assumption that embedding quality scales strictly with dimensionality. By activating dimensions from a pool, reweighting compact embeddings, or transforming static seeds, our architectures employ mixture-of-experts with soft routing to optimize information flow.

Across five benchmarks, our 64-dimensional dynamic embeddings consistently match or outperform dense 512-dimensional baselines. The remixing approach reduces parameters by 82.5% (12.32M vs 70.46M), while the direct generative method provides the highest inference speed. Furthermore, experts demonstrate interpretable specialization by routing ambiguous terms to distinct experts based on semantic context.

These findings indicate that learned routing offers stronger inductive bias than raw dimensional budget. Future work could explore hierarchical expert organization and multimodal settings. Ultimately, context-adaptive routing provides a viable path for creating parameter-efficient models that maintain high performance with reduced computational costs.

7 Limitations

While our results demonstrate the efficacy of context-aware routing, we acknowledge several limitations. **Scale:** Our experiments were conducted on models up to 70M parameters due to compute constraints. While the efficiency gains are promising, verifying whether these trends hold for large-scale foundation models (e.g., 7B+ parameters) remains future work. **Retrieval Trade-offs:** As noted in our semantic analysis, the remixing approach suffers a degradation in zero-shot retrieval tasks (STS). By prioritizing contextual dis-

ambiguation, the model sacrifices some capacity for static keyword matching, making it potentially less suitable for dense retrieval applications without fine-tuning. **Inference Overhead:** Although our method improves batched throughput, the router introduces a fixed computational cost per step. For extremely latency-sensitive, single-stream applications (batch size of 1), this routing overhead might offset the gains from reduced embedding dimensionality.

Acknowledgements

In accordance with the AI Writing Assistance Policy, we acknowledge the use of large language models (specifically Gemini) to assist with rephrasing text for clarity, condensing sections to meet page limits, and refining mathematical notation. All generated text was reviewed, verified, and edited by the human authors, who take full responsibility for the content.

References

Alexei Baevski and Michael Auli. 2018. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853*.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, and 1 others. 2022. Glam: Efficient scaling of language models with mixture-of-experts. *International Conference on Machine Learning*, pages 5547–5569.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

David Ha, Andrew Dai, and Quoc V Le. 2017. Hypernetworks. In *International Conference on Learning Representations*.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*.

Matt Mahoney. 2011. Large text compression benchmark. http://www.mattmahoney.net/dc/text.html.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Quan Ngoc Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. 2016. The lambada dataset: Word prediction requiring a broad discourse context. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534.

Emmanouil Antonios Platanios and et al. 2018. Contextual parameter generation for universal neural machine translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 425–435.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q8bert: Quantized 8bit bert. *arXiv preprint arXiv:1910.06188*.

Leslie N Smith and Nicholay Topin. 2019. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, pages 369–386. SPIE.

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