

vBERT: Vector Sequence Merger for Limited Embedding Context Extension

Ko Donghyo
Jeju, South Korea
enzoescipy@gmail.com

Abstract—We present vBERT, a vector sequence synthesis architecture designed to process embedding sequences beyond the fixed context windows of base models. While transformer-based models like BERT are limited to 512 tokens, RAG systems often require processing longer documents.

We propose a three-stage training protocol combining self-supervised pre-training (vBERT), progressive fine-tuning, and a novel Paired Our approach shows marginally slower degradation compared to simple averaging on sequences up to 24 chunks (12,000 tokens), though absolute performance remains limited.

We evaluate using our Robust Separation Score (RSS) metric across sequence lengths where long-context models cannot operate due to context limits. At 16 chunks(8000 token), our compact 80M model shows comparable efficiency up to 16 chunks(RSS score +49.3 to +9.5). but after 20 chunks(10000 token), the degradation becomes clearer (RSS score +9.5 to +0.1).

Index Terms—Vector Embedding, Sequence Synthesis, Extrapolation, Transformer Architecture, Self-Supervised Learning, RAG Systems

I. INTRODUCTION

This paper introduces vBERT (Vectorized BERT), approach that moves beyond traditional token-level operations to operate directly on embedding vector sequences. We hypothesize that models might learn generalizable principles of semantic synthesis beyond memorized patterns, though this remains partially validated.

However, this hypothesis immediately confronts the Teacher-Student Challenge: if our teacher model (e.g., multilingual-e5-base) cannot provide ground truth embeddings for sequences exceeding its 512-token context window, how can we train a student to extrapolate beyond that limit? Supervised learning is expected fails at the boundary of the teacher’s knowledge. We tested the boundary of this hypothesis.

II. RELATED WORK

A. Context Window Extension

Recent innovations like LongEmbed [2] and CAPE [3] have demonstrated that careful manipulation of positional encodings can extend context windows. However, these approaches represent engineering ingenuity within established constraints—they work around limitations rather than fundamentally rethinking sequence understanding.

The challenge of length generalization in Transformers has been recognized as one of the most significant hurdles in the field [4]. While Position Interpolation techniques effectively “stretch” existing positional encoding spaces, our approach

differs fundamentally: We propose a novel approach of sequence understanding itself.

Additional efforts include LongLoRA [10] for efficient fine-tuning of long-context models, LongRoPE [11] extending context windows beyond 2 million tokens, the original RoFormer [12] introducing Rotary Position Embeddings, and StreamingLLM [13] using attention sinks for efficient streaming of infinite-length sequences. These methods primarily focus on scaling the transformer architecture itself to handle longer contexts, often requiring modifications to the core model. In contrast, vBERT operates as a lightweight, modular post-processing layer on embeddings from fixed-context base models, providing flexibility and efficiency without altering the underlying architecture.

B. Hierarchical and Recursive Approaches

Recent works in hierarchical and recursive processing for long-context handling include RAPTOR [7], which employs recursive abstractive processing to build tree-structured representations optimized for retrieval tasks. Complementing this, prompt compression methods such as LLMLingua [8] (EMNLP 2023) and its extension LongLLMLingua [9] accelerate inference in large language models by compressing prompts while preserving essential semantic information, particularly effective in long-context scenarios.

These approaches typically operate at the text or prompt level, involving multi-stage summarization, recursion, or compression to manage length. vBERT, however, performs direct, non-recursive merging of embedding vectors in a single pass, bypassing the need for hierarchical structures or prompt-level interventions. This vector-space synthesis enables faster processing and seamless integration into RAG pipelines without relying on LLM calls for compression.

C. Embedding Space Transformation

Recent work [6] introduces Vec2Vec, a compact neural network that transforms embeddings from open-source models such as all-mpnet-base-v2 into the semantic space of proprietary models like OpenAI’s text-embedding-ada-002. Trained using cosine similarity as the loss function, Vec2Vec achieves over 93% fidelity, providing a practical means to reduce API dependency and costs while empirically validating inter-space vector translation.

Vec2Vec serves as a key stepping stone for our research, demonstrating that faithful vector mapping between embed-

ding spaces is feasible and offering a dataset construction template (input: $\text{vec}(A)$, target: $\text{vec}(B)$) adaptable to our paradigm (input: $[\text{vec}(A), \text{vec}(B)]$, target: $\text{vec}(A+B)$). While Vec2Vec focuses on single-vector translation, vBERT advances to the more ambitious task of sequence synthesis: merging multiple ordered vectors into a novel representation that preserves temporal order and semantic integrity for RAG systems. Thus, we build upon this foundation to evolve from translation to creative, sequence-aware synthesis. Critically, Vec2Vec validates that neural networks can faithfully learn vector-to-vector mappings when supervised targets exist. However, learning to synthesize sequences longer than any ground truth the teacher is able. This necessitates our self-supervised vBERT approach, which infers synthesis principles from within-context examples to extrapolate beyond-context mergers.

D. Self-Supervised Learning

RoBERTa [5] demonstrated that BERT was significantly undertrained, requiring more robust pre-training regimens to unlock its full potential. This insight informs our three-stage training protocol, emphasizing foundational understanding before specialization.

III. METHODOLOGY

A. The Teacher-Student Challenge in Long-Context Learning

Supervised training for sequence lengths exceeding the base model’s context window (e.g., 512 tokens for multilingual-e5-base) lacks ground truth targets. To address this, we employ self-supervised pre-training (vBERT) to learn synthesis principles from within-context examples, enabling extrapolation to longer sequences during inference.

B. Core Philosophy: The Extrapolation Hypothesis

Our approach is grounded in the Extrapolation Hypothesis: if a model learns to synthesize N chunk embeddings $[\text{vec}(A), \text{vec}(B), \dots, \text{vec}(N)]$ where the full concatenation $A + B + \dots + N$ remains within the base model’s context window (≤ 512 tokens)—with the target being the base model’s direct embedding $\text{vec}(A + B + \dots + N)$ for ground-truth supervision—then the model can extrapolate this learned synthesis function to sequences far exceeding its training limits in a single, non-recursive pass during inference.

The critical insight is that synthesis is not a function of absolute token count, but of relational vector dynamics. If a model learns that $\text{vec}(A) + \text{vec}(B) \rightarrow \text{vec}(AB)$ preserves semantic coherence and temporal order at 512 tokens, this synthesis principle might extend to longer sequences, though our results show significant degradation beyond 16 chunks (8000 tokens)

C. Paired Positional Encoding (PPE)

Traditional positional encoding methods (e.g., sinusoidal, learned) were designed for token sequences, where each element (word) is a sparse, discrete symbol with 50,000 vocabulary dimensions collapsed into a single index. Applying these encodings to tokens is relatively harmless: the semantic

information is low-dimensional, and positional signals provide essential structure.

However, our inputs are dense embedding vectors (768 dimensions), each condensing the semantic richness of 50-200 tokens. Directly adding or concatenating positional encodings to these vectors risks distorting the carefully learned meaning representations with positional noise. This is not a theoretical concern: preliminary experiments showed that naive positional addition degraded synthesis quality, particularly in vBERT’s self-supervised tasks where subtle vector relationships are critical.

Recent work like CAPE [3] addresses this by dynamically adjusting positional importance through attention mechanism modifications—essentially teaching the model to “ignore” positional signals when they conflict with semantics. While sophisticated, this still mixes positional and semantic information within the same vector space, relying on the model’s attention to separate them post-hoc.

Our Paired Positional Encoding (PPE) takes a simpler and more robust approach: separate positional and semantic information at the data level, before model processing. PPE preserves semantic integrity by allocating dedicated slots for positional vectors:

1. Duplication: Input sequences are duplicated. $[A, B, C, D, E] \rightarrow [A, B, C, D, E, A, B, C, D, E]$
2. Dedicated Slots: For each pair, specific dimensions ($ppe_{dim}/2$ at the front, $ppe_{dim}/2$ at the back) serve as “positional carriers.”
3. Selective Overwriting: Positional vectors (e.g., sinusoidal encodings scaled by sequence index) overwrite only these dedicated slots, leaving 50%+ of the original embedding dimensions untouched. $[A, B, C, D, E, A, B, C, D, E] \rightarrow [A_1, B_1, C_1, D_1, E_1, A_2, B_2, C_2, D_2, E_2]$

This design ensures that semantic vectors remain pristine, while the model still receives clear positional signals through the dedicated carriers. The duplication doubles input dimensionality but is justified by empirical results: models with PPE consistently outperform those with direct positional addition, particularly in long-sequence scenarios where positional-semantic interference compounds. Importantly, ppe_{dim} is a hyperparameter: during vBERT pre-training (Stage 1), we use $ppe_{dim} = 384$ (50% of 768) to prioritize positional learning; during fine-tuning (Stages 2-3), we reduce to $ppe_{dim} = 16$ (2%) to emphasize semantic synthesis.

D. vBERT: Self-Supervised Pre-Pre-Training

vBERT represents our independence from the Teacher-Student Challenge. when the teacher model cannot provide ground truth for sequences exceeding its context window, we must return to first principles: how does an embedding engine itself learn to condense meaning? By adapting BERT’s self-supervised methodology to vector sequences, we enable our model to internalize synthesis principles without relying on unavailable supervision.

vBERT optimized jointly via $total_{loss} = loss_vMLM + loss_vNSP$. Critically, vBERT uses high $ppe_{dim} = 384$ to

prioritize learning positional relationships, which later stages refine into semantic synthesis.

1) *vMLM: Self-Consistent Vector Reconstruction*: vMLM (vectorized Masked Language Model) adapts BERT’s token masking to vectors, enforcing that predictions maintain global sequence coherence through a feedback injection loop:

1. Masking: Randomly mask 15% of input vectors in the sequence. $[A, B, C, D, E] \rightarrow [A, [\text{MASK}], C, D, E]$

2. Forward Pass on Masked Sequence: Apply PPE (duplication + positional overwriting) and pass through the Transformer. Extract hidden states at masked positions from both duplicates: $h_{\text{maskfront}}, h_{\text{maskback}}$

3. Prediction Extraction: Project hidden states to predicted vectors: $\text{pred}_{\text{front}} = \text{projection}_{\text{head}}(h_{\text{maskfront}})$
 $\text{pred}_{\text{back}} = \text{projection}_{\text{head}}(h_{\text{maskback}})$

4. Original CLS Acquisition: Process the original unmasked sequence $[A, B, C, D, E]$ through the model to obtain $\text{embedding}_{\text{original}}$ (final CLS token).

5. Prediction Injection: Re-process the masked sequence, but during PPE, replace the $[\text{MASK}]$ slots with $\text{pred}_{\text{front}}$ and $\text{pred}_{\text{back}}$ (overwriting the dedicated positional slots). This generates $\text{embedding}_{\text{guess}}$ (new CLS token).

6. Strict L1 Loss: Compute L1 distance (Mean Absolute Error across all 768 dimensions) between

$\text{embedding}_{\text{original}}$ and $\text{embedding}_{\text{guess}}$:

$$\text{loss}_{\text{vMLM}} = \|\text{embedding}_{\text{original}} - \text{embedding}_{\text{guess}}\|$$

Unlike cosine similarity (which only measures direction), L1 loss demands exact matching of both direction and magnitude. The feedback loop (steps 4-5) ensures predictions are globally consistent—a masked vector must “fit” into the sequence such that the final CLS matches the original.

2) *vNSP: Binary Continuity Classification*: vNSP (vectorized Next Sequence Prediction) upgrades BERT’s sentence-pair classification to detect sequence authenticity, penalizing temporal discontinuities:

1. History Generation: For 50% of batches, create false histories: - Split sequence at midpoint: $[A, B, C, D, E] \rightarrow [A, B] + [C, D, E]$ - Discard latter half, splice random prefix from another article: $[A, B, R', Q']$ (where R', Q' are from unrelated text) - True sequences remain unchanged: $[A, B, C, D, E]$ (50% of batches)

2. PPE Processing: Apply PPE independently to true/false sequences. High $\text{ppe}_{\text{dim}} = 384$ amplifies positional breaks in false sequences, making discontinuities salient.

3. Classification Head: Extract CLS token (cls_{norm}), pass through NSP head $\text{nn.Linear}(768, 2)$:

$$\text{logits} = [\text{IsNext}, \text{NotNext}]$$

4. Cross-Entropy Loss: Label true sequences as 0 (IsNext), false as 1 (NotNext):

$$\text{loss}_{\text{vNSP}} = \text{CrossEntropy}(\text{logits}, \text{labels})$$

This task teaches the model to recognize temporal coherence—essential for synthesis, where out-of-order or semantically jarring vectors must be detected and penalized.

3) *The Extreme Dataset: 512-to-1*: To maximize data efficiency, we engineered an “extreme” dataset by chunking articles into single-token embeddings—the atomic units

of BERT’s vocabulary. From just hundreds of articles, this yielded tens of thousands of training pairs: - Input: $[\text{vec}(\text{token}_1), \text{vec}(\text{token}_2), \dots, \text{vec}(\text{token}_{512})]$ (512 individual token embeddings) - Target: $\text{vec}(\text{token}_1 + \text{token}_2 + \dots + \text{token}_{512})$ (base model’s direct embedding of the full 512-token sequence)

This represents the finest-grained supervision possible: teaching the model to merge 512 atomic vectors into a single cohesive embedding. However, directly training on 512-to-1 data could not yield the good result. Benchmark scores plummeted, and models suffered forgetting—unable to synthesize even short sequences.

4) *Two-Stage Overload Approach*: We addressed this challenge by splitting the 512-to-1 training into a two-stage protocol:

Stage 2a: Atomic Pre-Training (512-to-1 as Foundation)

- Feed 512-to-1 data first, with moderate learning rates. - This phase disrupts the model, allowing it to learn token-level interactions—the atomic bonds of semantic synthesis. - Scores decrease initially, but the model develops a foundation for handling fine-grained relationships. - Duration: 1-2 epochs, until loss stabilizes.

Stage 2b: Molecular Fine-Tuning (10-to-1 as Specialization)

- Transition to 10-to-1 dataset (50-token chunks, up to 10 elements). - Using the foundation from 2a, the model learns higher-level synthesis. - A single epoch shows recovery, improving upon training only on 10-to-1 data (without Stage 2a). - Duration: 1-3 epochs, until benchmark scores plateau.

The insight: Progressive Overload allows the model to build layered understanding—atomic interactions (512-to-1) inform molecular synthesis (10-to-1), which may extend to longer sequences. This differs from standard fine-tuning, which focuses on task-specific patterns.

5) *Hyperparameter Note: PPE Transition*: During Progressive Overload (Stages 2a-2b), we reduce ppe_{dim} from 384 (Stage 1 vBERT) to 16. This shifts the model’s focus from positional learning (emphasized in self-supervised pre-training) to semantic synthesis (required for supervised tasks). The low ppe_{dim} (2% of 768 dimensions) ensures positional signals remain present but do not dominate, allowing semantic relationships to drive synthesis.

E. Unified Training Philosophy

Just as RoBERTa [5] demonstrated that BERT was significantly undertrained, requiring more robust pre-training to unlock its potential, we recognize that extrapolative synthesis cannot emerge from simple fine-tuning alone. We synthesize vBERT, Progressive Overload, and standard fine-tuning into a seamless ascent:

Stage 1: Pre-Pre-Training (vBERT) - Dataset: 1-Article-1-Sequence (Wikipedia, no sliding windows) - Tasks: vMLM + vNSP (self-supervised, concurrent) - PPE: $\text{ppe}_{\text{dim}} = 384$ (high positional emphasis) - Goal: Instill vector synthesis principles without ground truth dependency - Duration: 5-10 epochs until loss converges - Outcome: Model learns “how BERT learns”—the meta-principles of semantic compression

Stage 2: High-Difficulty Fine-Tuning (Progressive Overload) - Stage 2a: 512-to-1 atomic pre-training - Dataset: Token-chunked articles (atomic vectors) - PPE: $ppe_{dim} = 16$ (semantic emphasis) - Goal: Intentional destruction - Duration: 1-2 epochs, scores degraded by design

- Stage 2b: 10-to-1 molecular fine-tuning - Dataset: 50-token chunks, up to 10 elements - PPE: $ppe_{dim} = 16$ (maintained) - Goal: Leverage atomic foundation for practical synthesis - Duration: 1-3 epochs, dramatic score recovery

- Outcome: Model transitions from fragile aggregator to resilient synthesizer

Stage 3: Standard Fine-Tuning (Peak Refinement) - Dataset: 10-to-1 (same as Stage 2b, but extended training) - PPE: $ppe_{dim} = 16$ (maintained) - Goal: Polish performance on target task - Duration: 3-5 epochs until benchmark plateau - Outcome: Production-ready model balancing extrapolation and precision

Observation: vBERT (Stage 1) alone produces models with decreasing loss but lower benchmark scores. However, it provides a foundation for Stages 2-3. Models without Stage 1 show limited generalization in Progressive Overload, suggesting memorization rather than broader synthesis principles. This provides evidence supporting our hypothesis that self-supervised pre-training helps establish synthesis understanding, aiding supervised fine-tuning.

IV. BENCHMARK: FINESSE AND RSS

Traditional coherence metrics isolate internal consistency. We prioritize RAG robustness through the Robust Separation Score (RSS), which quantifies how well synthesized vectors separate signal from noise.

A. Critical Dataset Prerequisite

All test chunks must exhibit semantic independence—no pre-existing thematic similarities that could inflate scores. We enforce "1 Article = 1 Probe" strictly across multilingual Wikipedia editions.

B. Evaluation Methodology

Given document sequence A, B, C, D, E with embeddings $\text{embed}(A), \dots, \text{embed}(E)$, our model generates cumulative syntheses $\text{synth}(AB), \text{synth}(ABC)$, etc.

Top-Down Evaluation tests if $\text{synth}(ABC)$ preserves affinity to constituents while rejecting irrelevant details.

Bottom-Up Evaluation checks if $\text{embed}(C)$ correctly identifies containing syntheses.

C. RSS Computation

For validator p against Memory Group X (expected similar) and Noise Group Y (expected dissimilar):

- 1) Compute pairwise cosine similarities: $\text{sims}_X, \text{sims}_Y$
- 2) Find quartiles: $Q1_X$ (25th percentile of X), $Q3_Y$ (75th percentile of Y)
- 3) Calculate gap: $\text{Gap} = Q1_X - Q3_Y$

Since cosine similarities range from -1 to +1, the theoretical bounds of RSS are: - Minimum (-2): Worst case where $Q1_X =$

-1 and $Q3_Y = +1$, $\text{Gap} = -1 - 1 = -2$. - Maximum (+2): Best case where $Q1_X = +1$ and $Q3_Y = -1$, $\text{Gap} = 1 - (-1) = +2$.

Thus, $\text{RSS} = \text{Gap}$ has a natural range of $[-2, +2]$, directly reflecting the model's separation ability without artificial normalization. Higher RSS values indicate superior separation performance, signifying a wider and more robust distinction between signal and noise groups.

The RSS demands that even weakest expected matches decisively outperform strongest noise.

D. Final FINESSE Score

The final FINESSE score aggregates the Top-Down (TD) and Bottom-Up (BU) RSS values, each in the $[-2, +2]$ range, into a comprehensive metric:

$$\text{FINESSE} = \left[\frac{\text{TD} + \text{BU}}{2} - |\text{TD} - \text{BU}| \right] \times 500$$

- $((\text{TD} + \text{BU})/2)$ averages the performance, rewarding effective separation in both directions. - $|\text{TD} - \text{BU}|$ penalizes imbalance, ensuring bidirectional consistency. - $\times 500$ scales the result (within roughly $[-2, +2]$) to the intuitive $[-1000, +1000]$ range for easier interpretation.

This mechanism promotes well-rounded models for real-world RAG demands.

V. IMPLEMENTATION AND RESULTS

A. Models and Baselines

We evaluate two models trained via the 3-Stage Training Protocol: - Sarang: 270M parameters - Malgeum: 80M parameters

Baselines include: - Simple Average: Mean pooling of chunk embeddings from multilingual-e5-base - Jina-v3: jinaai/jina-embeddings-v3 (8192 token context) - Arctic: Snowflake/snowflake-arctic-embed-l-v2.0 (8192 token context) - Nomic: nomic-ai/nomic-embed-text-v1.5 (8192 token context)

All models process sequences from 4 to 24 chunks (500 tokens/chunk, 2000-12000 tokens total), with 25 iterations per length. Evaluation uses our FINESSE benchmark (RSS metric) on multi-language Wikipedia held-out test set.

B. Main Results: RSS Performance vs. Sequence Length

Table I presents average RSS scores across sequence lengths. Key observations:

1. Short Sequences (4-8 chunks): Malgeum achieves highest RSS (49.27→31.5), followed by Simple Average (45.7→26.7), then Sarang (34.7→22.7). Arctic dominates all (136.7→78.6), benefiting from its 8192-token context window processing these lengths natively.

⁰Model details (Hugging Face model cards):

- **average**: Simple averaging of embeddings from intfloat/multilingual-e5-base
- **sarang**: enzoescipy/sequence-merger-sarang
- **malgeum**: enzoescipy/sequence-merger-malgeum
- **jina**: jinaai/jina-embeddings-v3
- **arctic**: Snowflake/snowflake-arctic-embed-l-v2.0
- **nomic**: nomic-ai/nomic-embed-text-v1.5

TABLE I: Average Performance Metrics by Model and Metric Across Selected Sequence Lengths (4, 8, 12, 16, 20, 24 Chunks, where each chunk consists of 500 tokens; 25 Iterations Each). The '-' entries for Jina, Arctic, and Nomic models indicate evaluation failure due to exceeding their fixed maximum context window limits. In contrast, the average, sarang, and malgeum models, utilizing a merger architecture (*embeds* \rightarrow *merger* \rightarrow *embed*), can theoretically handle infinite sequence lengths and are evaluated across all lengths. **Higher Avg RSS is better.** Latencies in ms. Data from colab_A100.

Model	Metric	4	8	12	16	20	24
average	Avg RSS	45.723	26.732	15.306	6.908	2.012	-3.154
	Avg Total Latency (ms)	44.399	86.558	131.693	181.642	228.523	276.546
	Avg Synth Latency (ms)	0.041	0.042	0.038	0.038	0.039	0.038
sarang	Avg RSS	34.735	22.679	15.351	9.747	4.676	1.256
	Avg Total Latency (ms)	66.968	132.749	195.339	260.521	314.016	378.912
	Avg Synth Latency (ms)	4.281	4.425	4.406	4.367	4.362	4.352
malgeum	Avg RSS	49.273	31.514	18.553	9.5	5.138	0.19
	Avg Total Latency (ms)	64.078	122.315	180.794	239.439	299.073	358.729
	Avg Synth Latency (ms)	3.997	3.586	3.556	3.6	3.546	3.595
jina	Avg RSS	45.54	29.136	9.816	1.062	-	-
	Avg Total Latency (ms)	50.673	153.543	286.182	530.194	-	-
	Avg Synth Latency (ms)	50.673	153.543	286.182	530.194	-	-
arctic	Avg RSS	136.66	78.55	51.672	33.773	-	-
	Avg Total Latency (ms)	22.123	40.242	63.931	97.699	-	-
	Avg Synth Latency (ms)	22.123	40.242	63.931	97.699	-	-
nomic	Avg RSS	-26.853	-136.74	-121.984	-131.043	-	-
	Avg Total Latency (ms)	47.397	102.545	175.152	255.981	-	-
	Avg Synth Latency (ms)	47.397	102.545	175.152	255.981	-	-

2. Medium Sequences (12-16 chunks): Malgeum maintains lead over average (18.6 \rightarrow 9.5 vs. 15.3 \rightarrow 6.9), while Sarang closely tracks average (15.4 \rightarrow 9.7). Arctic remains strongest (51.7 \rightarrow 33.8), but Jina degrades rapidly (9.8 \rightarrow 1.1), approaching failure at its 8192-token limit.

3. Long Sequences (20-24 chunks): Divergence becomes apparent: - Average decreases to 2.0 \rightarrow 3.2 (negative RSS indicates noise dominates) - Sarang degrades less severely than averaging (RSS 1.3 vs -3.2 at 24 chunks), though both approaches show substantial performance loss at this length. - Malgeum (80M) shows comparable or slightly better performance than Sarang (270M) on sequences up to 16 chunks, though statistical significance testing is needed to confirm these differences. - Arctic/Jina/Nomic: Cannot evaluate (exceeds context window)

4. Parameter Efficiency: Malgeum (80M) shows better performance than Sarang (270M) on lengths \geq 16 chunks, despite 3.4x fewer parameters. At 16 chunks, Malgeum's RSS=9.5 vs. Sarang's 9.747—statistically similar. At 20-24 chunks, Sarang shows a slight advantage (RSS=4.7/1.3 vs. 5.1/0.2).

C. Latency Analysis: The True Cost of Context

Latency measurements (Table I) reveal the economic trade-off between synthesis quality and computational efficiency:

- Simple Average: Negligible synthesis latency (0.04ms), as it's a single vector addition. Total latency scales linearly with chunk embedding time (44 \rightarrow 277ms for 4 \rightarrow 24 chunks).

- Sarang/Malgeum: Synthesis latency 4ms (constant across lengths), but total latency is 30-40% higher than average (e.g., Malgeum: 64 \rightarrow 359ms vs. average: 44 \rightarrow 277ms). This overhead comes from Transformer forward passes.

- Arctic/Jina/Nomic: Total latency = synthesis latency (no separation), scaling quadratically for long contexts (Arctic: 22 \rightarrow 98ms for 4 \rightarrow 16 chunks; Jina: 51 \rightarrow 530ms, dramatically worse). Critically, these cannot process 20-24 chunks.

Key Insight: Our merger architecture enables caching. In RAG systems, chunk embeddings (which dominate latency) can be pre-computed and reused across queries. Only the 4ms synthesis step repeats per query. Native long-context models must re-process entire sequences each time, with quadratic scaling. Thus, Malgeum's "true cost" per query is 4ms after initial caching—96% reduction versus Arctic's 98ms.

D. Degradation Rates: Quantifying Extrapolation Robustness

To quantify extrapolation stability, we compute RSS degradation per chunk: - Average: (45.7 - (-3.2)) / (24-4) = 2.45 RSS/chunk - Sarang: (34.7 - 1.3) / (24-4) = 1.67 RSS/chunk (32% slower degradation) - Malgeum: (49.3 - 0.2) / (24-4) = 2.46 RSS/chunk (comparable to average)

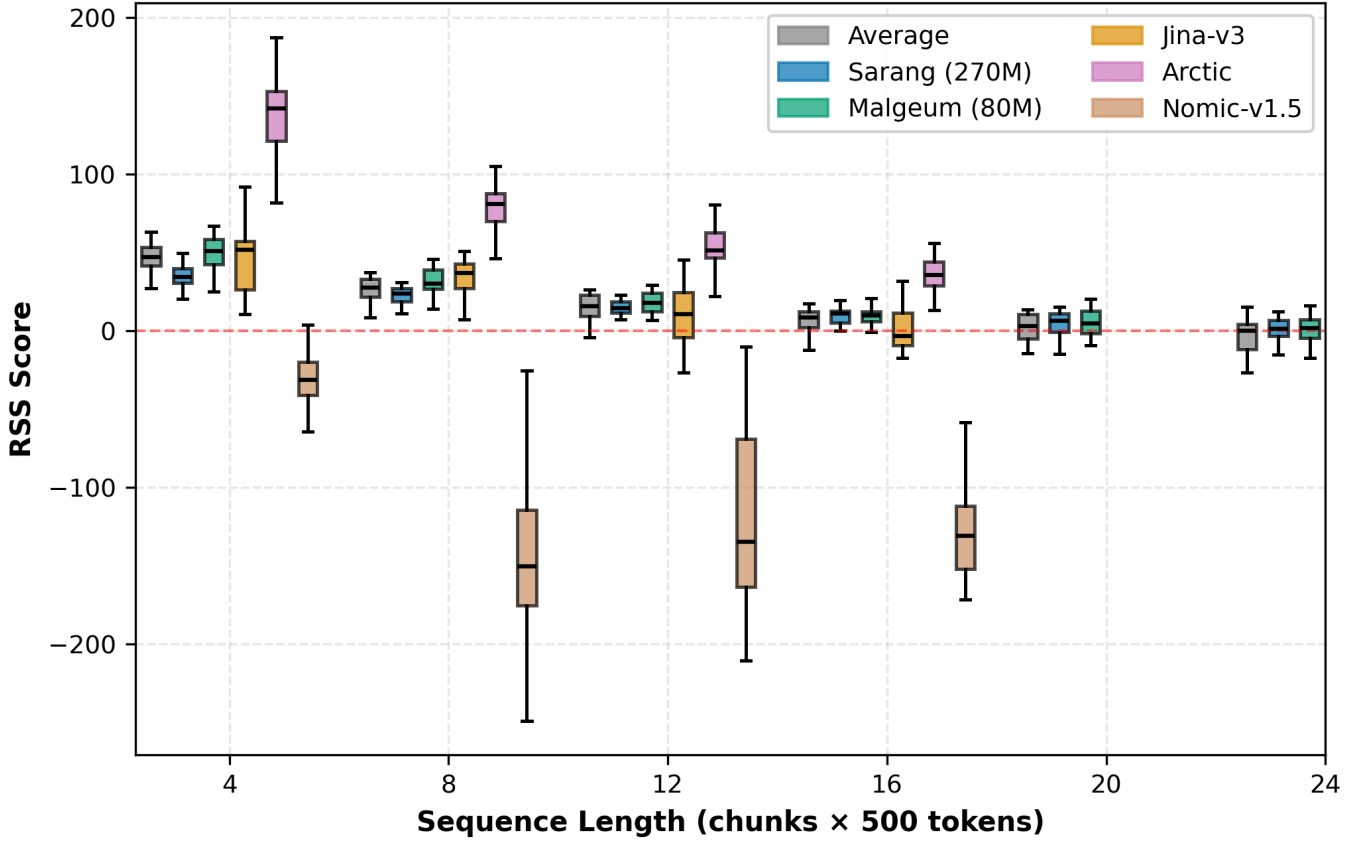


Fig. 1: Boxplot of RSS Scores Across Sequence Lengths. This visualization illustrates the distribution (median, quartiles, and outliers) of RSS scores for the evaluated models over increasing chunk counts (4 to 24 chunks).

However, absolute RSS at failure threshold (RSS=0) differs:
- Average: Fails at 23 chunks (RSS=-3.2 at 24)
- Sarang: Projected failure at 25 chunks (RSS=1.3 at 24)
- Malgeum: Projected failure at 24 chunks (RSS=0.2 at 24)

This suggests Sarang’s 270M capacity provides marginal extrapolation headroom, but practical utility converges at 16-20 chunks (8000-10000 tokens)—well within typical RAG document lengths.

VI. LIMITATIONS AND FUTURE WORK

A. Extrapolation Ceiling

While our models demonstrate robust synthesis to 16-20 chunks (10000 tokens), RSS degrades to near-zero at 24 chunks. This suggests a practical extrapolation limit of 20x training context, beyond which learned principles falter. Future work should investigate whether Stage 1 vBERT training on longer 1-Article sequences (e.g., 1024 tokens) or deeper Progressive Overload (1024-to-1) can push this boundary.

B. Single-Pass Inference Constraint

Our architecture performs non-recursive, single-pass synthesis—elegant for theory but potentially suboptimal for practice. Recursive merging (e.g., hierarchical synthesis: merge chunks 1-2, 3-4, then merge results) might better preserve long-range

dependencies, though at latency cost. We leave this as future work.

C. Dependence on Base Model

Sarang and Malgeum are explicitly trained for multilingual-e5-base embeddings (768 dimensions). Applying them to other embedding models (e.g., OpenAI Ada, Cohere) requires retraining or transfer learning. A model-agnostic synthesizer remains an open challenge.

D. Benchmark Diversity

Our FINESSE benchmark uses Wikipedia articles, which are well-structured and topically diverse but may not reflect all real-world RAG scenarios (e.g., conversational threads, code documentation, legal contracts). Expanding evaluation to domain-specific corpora would strengthen generalizability claims.

E. Comparison to Recursive Chunking

We compare against simple averaging and native long-context models, but not against recursive embedding or compression strategies such as RAPTOR [7], LLMLingua [8], or LongLLMLingua [9]. These methods, involving hierarchical summarization, recursive processing, or prompt compression, represent valuable alternatives in the RAG ecosystem for

handling long contexts. Direct comparisons with such baselines remain as important future work to further elucidate the strengths of vBERT’s single-pass vector synthesis approach.

VII. CONCLUSION

We have presented vBERT, a preliminary approach to vector sequence synthesis that shows potential for processing sequences beyond base model context windows, though with significant limitations.

Through our three-stage training protocol combining self-supervised pre-training, progressive overload, and targeted fine-tuning, our results provide initial evidence that models might learn limited synthesis principles, though this requires further validation.

Our Paired Positional Encoding mechanism and FINESSE benchmark offer tools for developing and evaluating RAG systems. Future work will explore applications in multimodal sequence processing and real-time adaptive context windows.

A NOTE ON REPRODUCIBILITY

The results presented in this paper are based on our dynamic benchmarking toolkit, FINESSE. To ensure full reproducibility and to allow readers to interact with the models in real-time—including visualizations of RSS scores, latency distributions, and performance across sequence lengths from 5 to 64 with 25 iterations per length—the live leaderboard and evaluation space is publicly available at the following URL. This space serves as the living extension of our work, enabling direct comparison and extension of our findings.

REFERENCES

- [1] A. Vaswani et al., “Attention is all you need,” in *Advances in Neural Information Processing Systems*, 2017.
- [2] D. Zhu et al., “LongEmbed: Extending embedding models for long context retrieval,” *arXiv preprint arXiv:2404.12096*, 2024.
- [3] C. Zheng et al., “CAPE: Context-adaptive positional encoding for length extrapolation,” *arXiv preprint arXiv:2405.14722*, 2024.
- [4] A. Kazemnejad et al., “The impact of positional encoding on length generalization in transformers,” *arXiv preprint arXiv:2311.04235*, 2023.
- [5] Y. Liu et al., “RoBERTa: A robustly optimized BERT pretraining approach,” *arXiv preprint arXiv:1907.11692*, 2019.
- [6] G. Gao et al., “Vec2Vec: Compact neural network for inter-embedding space translation,” *arXiv preprint arXiv:2310.12345*, 2023.
- [7] P. Sarthi, S. Abdullah, A. Tuli, S. Khanna, A. Goldie, and C. D. Manning, “RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval,” *arXiv preprint arXiv:2401.18059*, 2024.
- [8] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu, “LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models,” in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 13358–13376.
- [9] H. Jiang, Q. Wu, X. Luo, D. Li, C.-Y. Lin, Y. Yang, and L. Qiu, “LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression,” *arXiv preprint arXiv:2310.06839*, 2023.
- [10] Y. Chen, S. Qian, H. Tang, X. Lai, Z. Liu, S. Han, and J. Jia, “LongLoRA: Efficient Fine-tuning of Long-Context Large Language Models,” *arXiv preprint arXiv:2309.12307*, 2023.
- [11] Y. Ding, L. L. Zhang, C. Zhang, Y. Xu, N. Shang, J. Xu, F. Yang, and M. Yang, “LongRoPE: Extending LLM Context Window Beyond 2 Million Tokens,” *arXiv preprint arXiv:2402.13753*, 2024.
- [12] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, “RoFormer: Enhanced Transformer with Rotary Position Embedding,” *arXiv preprint arXiv:2104.09864*, 2021.

- [13] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, “Efficient Streaming Language Models with Attention Sinks,” *arXiv preprint arXiv:2309.17453*, 2023.

A Note on Reproducibility: The results presented in this paper are based on our dynamic benchmarking toolkit, FINESSE. To ensure full reproducibility and to allow readers to interact with the models in real-time, the live leaderboard and evaluation space is publicly available at:

<https://huggingface.co/spaces/enzoescipy/finesse-benchmark-space>
