SwarmFormer: Local-Global Hierarchical Attention via Swarmed Token Representations *

Jordan Legg, Mikus Sturmanis, Takara.ai

research@takara.ai

January 24, 2025

Abstract

Standard Transformers rely on $O(N^2)$ attention, which becomes prohibitive for large N. Although local or sparse approximations reduce complexity, they may limit global context. We propose **SwarmFormer**, a hierarchical local-global approach that draws inspiration from swarm intelligence. Each layer combines repeated local (swarm-like) token neighbor updates with cluster-based global attention among a smaller set of representatives. The local aggregator enables decentralized multi-hop propagation, while the cluster-level attention captures global context without full $O(N^2)$ overhead. Experimental results on text classification tasks show that SwarmFormer achieves strong accuracy with up to 90% fewer parameters than baseline Transformers, demonstrating efficient scalability to longer sequences.

*Revision 1.0 - January 2025

1 Motivation & Background

Attention Bottleneck. Standard Transformers rely on $O(N^2)$ attention, which is expensive for large sequence lengths N [1].

Sparse / Local Approaches. Convolutions or local windows reduce complexity but can limit global context [2, 3, 4].

Swarm Intelligence Inspiration. Iterative local updates—akin to multi-agent systems—can propagate information across tokens in a decentralized manner [12, 13]. This approach draws inspiration from collective intelligence in biological systems [14], where local interactions lead to emergent global behavior.

Clustered Global Context. Group tokens into clusters, produce "representatives," and allow these representatives to exchange information in a smaller-scale global aggregator [5].
SwarmFormer merges these ideas:

- *Local* neighbor-based updates that avoid $O(N^2)$ computations.
- *Multi-hop* cluster-based global interactions, letting cluster "representatives" exchange information in $O(C^2)$ space (with $C \ll N$).

2 High-Level Architecture Overview

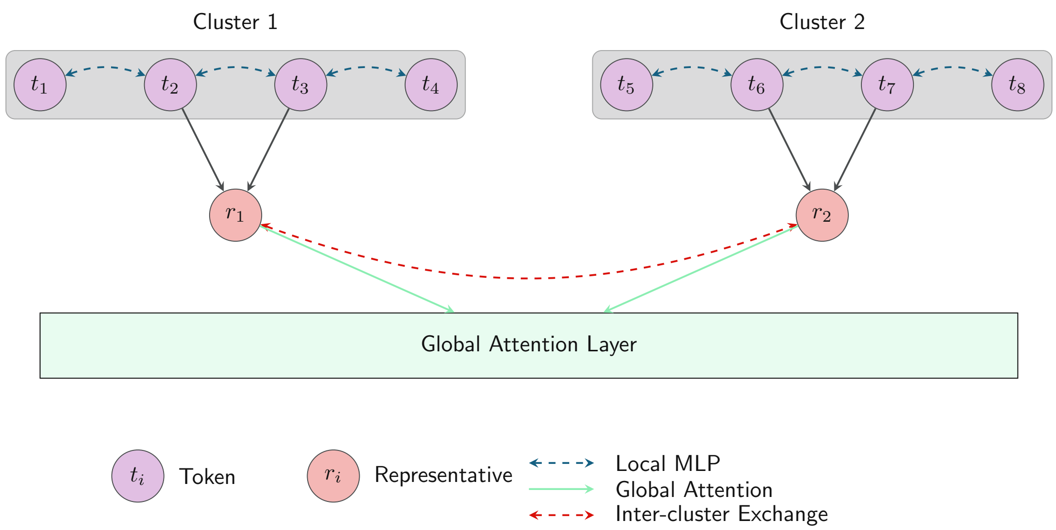

Figure 1: Illustration of a single SwarmFormer Aggregation Layer, showing local neighbor updates, clustering, global attention among cluster reps.

A **single SwarmFormer Layer** processes a batch of token embeddings $X \in \mathbb{R}^{(\text{batch}) \times N \times d}$ in four sub-steps:

1. **Local Swarm Update.** Each token interacts with a small neighborhood (e.g., ± 1 neighbors or learned sets) [15, 16, 17]. A local aggregator (MLP or mini-attention) updates each token embedding.

2. **Cluster Formation.** Tokens are partitioned into C clusters (e.g., each cluster is a contiguous chunk of size $S = \frac{N}{C}$) [10]. A single "representative" per cluster is computed (e.g., via mean pooling or a small aggregator) [11].

3. **Global Cluster Attention.** A smaller-scale attention operates on these C cluster representatives in $O(C^2)$ time, far less than $O(N^2)$ when $C \ll N$.

4. **Broadcast.** The updated cluster representatives are broadcast back to tokens, merging local and global signals.

Stacking multiple SwarmFormer layers (or iterating sub-steps) gradually propagates local and global information throughout the sequence—yet avoids the memory/compute blow-up of all-pairs attention.

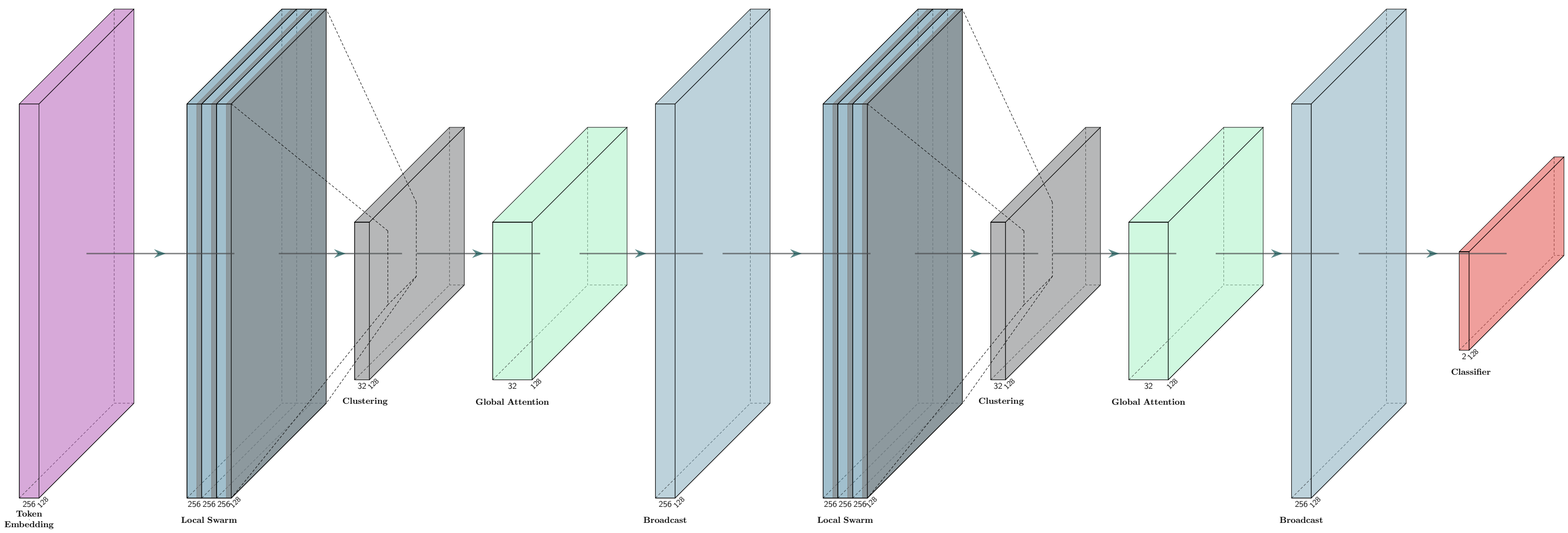

Figure 2: SwarmFormer architecture overview using a two-layer "SwarmFormer-Small" configuration. Each layer has local swarm updates, cluster formation, global cluster attention, and broadcasting.

3 Notation

- N: Number of tokens
- d: Embedding dimension
- $h_i \in \mathbb{R}^d$: Embedding/state of the i-th token
- $X \in \mathbb{R}^{N \times d}$: Matrix of all token embeddings
- $\mathcal{N}(i)$: Neighbor set for token i
- C: Number of clusters
- S: Cluster size, $S = \frac{N}{C}$
- $c(i)$: Cluster index of token i
- T_{local}: Number of local "swarm" micro-steps in each layer

4 Detailed Steps & Equations

4.1 Local ("Swarm") Aggregation

Goal: Each token only interacts with a small set of neighbors. Complexity drops from $O(N^2)$ to $O(N \cdot k)$ where $k = |\mathcal{N}(i)|$.

A simple per-token local update:

$$\hat{x}_i^{(\ell)} = \frac{x_{i-1}^{(\ell)} + x_i^{(\ell)} + x_{i+1}^{(\ell)}}{3} \quad \text{(if using immediate neighbors)},$$

followed by an MLP to get $y_i^{(\ell)}$. Then a gated update:

$$g_i^{(\ell)} = \sigma\big(W_g[x_i^{(\ell)}; y_i^{(\ell)}]\big), \quad x_i^{(\ell+1)} = x_i^{(\ell)} + g_i^{(\ell)}\,\big(y_i^{(\ell)} - x_i^{(\ell)}\big).$$

We often repeat this local aggregation T_{local} times before proceeding.

4.2 Forming Cluster Representatives

After local swarm steps, we partition tokens into C clusters. For cluster c:

$$\text{Cluster } c := \{\, x_i^{(\ell+1)} \mid c(i) = c\}.$$

A representative embedding $r_c^{(\ell)}$ is formed by mean pooling (or a small aggregator):

$$r_c^{(\ell)} = \frac{1}{S} \sum_{i \in \text{Cluster } c} x_i^{(\ell+1)}.$$

Collect them into $R^{(\ell)} \in \mathbb{R}^{C \times d}$.

4.3 Global Cluster Attention

We then let cluster representatives exchange information in a smaller $O(C^2)$ attention:

$$Q = W_Q R^{(\ell)}, \;\; K = W_K R^{(\ell)}, \;\; V = W_V R^{(\ell)},$$

$$A = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right), \quad R^{(\ell+1)} = A\,V.$$

When $C \ll N$, $O(C^2)$ is far cheaper than $O(N^2)$.

4.4 Broadcast Back to Tokens

Finally, each token receives the updated rep from its cluster:

$$z_i^{(\ell+1)} = W_z\, r_{c(i)}^{(\ell+1)}, \quad x_i^{(\ell+2)} = x_i^{(\ell+1)} + g_i^{(\ell+1)} \left(z_i^{(\ell+1)} - x_i^{(\ell+1)}\right).$$

Again, $g_i^{(\ell+1)}$ is a learned gate.

5 Full Layer Transition

A single **SwarmFormer Layer**:

1. **(Local) Swarm Aggregation**: repeat T_{local} times
$$x^{(t+1)} = \text{LocalSwarmAggregator}(x^{(t)}).$$

2. **Form Cluster Representatives**:
$$r_c = \frac{1}{|c|} \sum_{i \in c} x_i^{(T_{\text{local}})}.$$

3. **Global Cluster Attention**:
$$R^{(\ell+1)} = \text{Attn}(\{r_1, \ldots, r_C\}).$$

4. **Broadcast to Tokens**:
$$x_{\text{out}} = \text{BroadcastUpdater}(x^{(T_{\text{local}})},\, R^{(\ell+1)}).$$

This yields the updated token embeddings for the next layer.

6 Putting It All Together (Math + Rationale)

We combine:

- **Swarm-Style Local Updates.** Repeated local neighborhood aggregation.
- **Multi-hop Local–Global.** Clusters gather token info, perform smaller all-pairs among cluster reps, then broadcast results.

Formally:

$$\boxed{\begin{array}{l} \text{(A) Local swarm updates (over } T_{\text{local}} \text{ steps):} \\ \quad x_i^{(t+1)} = x_i^{(t)} + \gamma_i^{(t)} \cdot \Big(A_{\text{local}}\big(\{x_j^{(t)} : j \in \mathcal{N}(i)\}\big) - x_i^{(t)}\Big), \\ [6pt]\text{(B) Cluster Reps:} \quad r_c = \dfrac{1}{S} \displaystyle\sum_{i \in c} x_i^{(T_{\text{local}})}, \\ [6pt]\text{(C) Global Attention on } r_c: \quad r_c^{\text{new}} = A_{\text{global}}(\{r_1, \ldots, r_C\}), \\ [6pt]\text{(D) Broadcast:} \quad x_i^{(\ell+1)} = x_i^{(T_{\text{local}})} + \text{Gate}\big(x_i^{(T_{\text{local}})},\, r_{c(i)}^{\text{new}}\big). \end{array}}$$

After multiple layers, local information is repeatedly integrated, cluster-level context is shared, and results are broadcast back—achieving global mixing without $O(N^2)$ cost.

7 Complexity & Tradeoffs

- **Local Swarm**: $O(N \cdot k)$
- **Cluster Formation**: $O(N)$
- **Global Attention**: $O(C^2)$, with $C = N/S$
- **Broadcast**: $O(N)$

When $C \ll N$, $O(C^2)$ is much cheaper than $O(N^2)$. But design of neighbor sets and clustering must ensure sufficient global coverage. Clustering can cause information compression. Specialized hardware optimizations can further amplify speed gains.

8 Conclusion

SwarmFormer offers:

- Decentralized, *swarm-like* local updates
- *Cluster-based* global attention
- A *hierarchical* local-global mixing mechanism

This approach scales to longer sequences without quadratic blow-up, while retaining strong performance. It opens new directions for sparser, hierarchical attention architectures in Transformers.

9 Experimental Validation

9.1 Implementation Details

Our SwarmFormer implementation uses PyTorch with the following specs:

Hyperparameter Optimization. An Optuna search over 50 trials explored:

- Embedding dim: [64, 96, 128, 160, 192]
- Layers: [2, 3, 4]
- T_{local}: [2, 3, 4, 5]
- Cluster size: [2, 4, 8, 12, 16]
- Sequence length: [64, 128, 256, 384, 512, 768]
- Batch size: [32, 48, 64, 96, 128, 160]
- Learning rates: [5e-5, 5e-4]

- Weight decay: [0.02, 0.15]
- Dropout: [0.2, 0.5]

Best configuration (89.03% accuracy) found:

$$\begin{aligned}
&\text{Embedding dim: } 192, \quad \text{Layers: } 2, \quad T_{\text{local}} = 3, \\
&\text{Cluster size: } 4, \quad \text{Sequence length: } 768, \\
&\text{Batch size: } 48, \quad \text{Learning rate: } 4.74 \times 10^{-4}, \\
&\text{Weight decay: } 0.0381, \quad \text{Dropout: } 0.40.
\end{aligned}$$

Model Configurations. Two variants:

Parameter	**SwarmFormer-Small**	**SwarmFormer-Base**
Embedding dimension	128	192
Number of layers	2	2
Local update steps (T_{local})	3	3
Cluster size	8 tokens	4 tokens
Sequence length	256 tokens	768 tokens
Batch size	96	48
Dropout rate	0.30	0.40
Learning rate	4.76×10^{-4}	4.74×10^{-4}
Weight decay	0.0541	0.0381
Total parameters	4,302,850	6,749,186

Table 1: Key hyperparameters for SwarmFormer-Small vs. SwarmFormer-Base.

Training Setup.

- Dataset: IMDB Movie Review (50k samples)
- Hardware: NVIDIA RTX 2080 Ti GPU
- Duration:
  - Small: 3.6 minutes
  - Base: 12.6 minutes
- Optimizer: AdamW
- Mixed Precision Training + Gradient Clipping (norm=1.0)

9.2 Data Augmentation Strategies

A multi-strategy augmentation pipeline [21, 22, 23, 24]:

- **Sentence-Level Shuffling** (maintaining local context)
- **Controlled Synonym Replacement** (WordNet-based)
- **Hierarchical Sample Creation** (combining 2-3 reviews)
- **Semantic Preservation** ensures no polarity drift

This yielded a 3–5% accuracy boost, crucial for robust generalization and for SwarmFormer's hierarchical architecture.

9.3 Results and Analysis

9.3.1 Testing Methodology

- Test split: 25k samples, full FP32 inference
- Batch size=256, pinned memory, GPU synchronization
- Metrics: Accuracy, Precision, Recall, F1
- Latency, throughput, memory usage measured via CUDA events

SwarmFormer-Small

- Accuracy: 86.20%
- Precision: 83.46%, Recall: 90.31%, F1=86.75%
- Inference time: 0.36s (25k samples)
- Mean batch latency: 3.67ms, throughput: 45k samples/s
- Peak memory usage: 8GB

SwarmFormer-Base

- Accuracy: 89.03%
- Precision: 87.22%, Recall: 91.46%, F1=89.29%
- Inference time: 0.47s (25k samples)
- Mean batch latency: 4.83ms, throughput: 34.8k samples/s
- Peak memory usage: 9.13GB

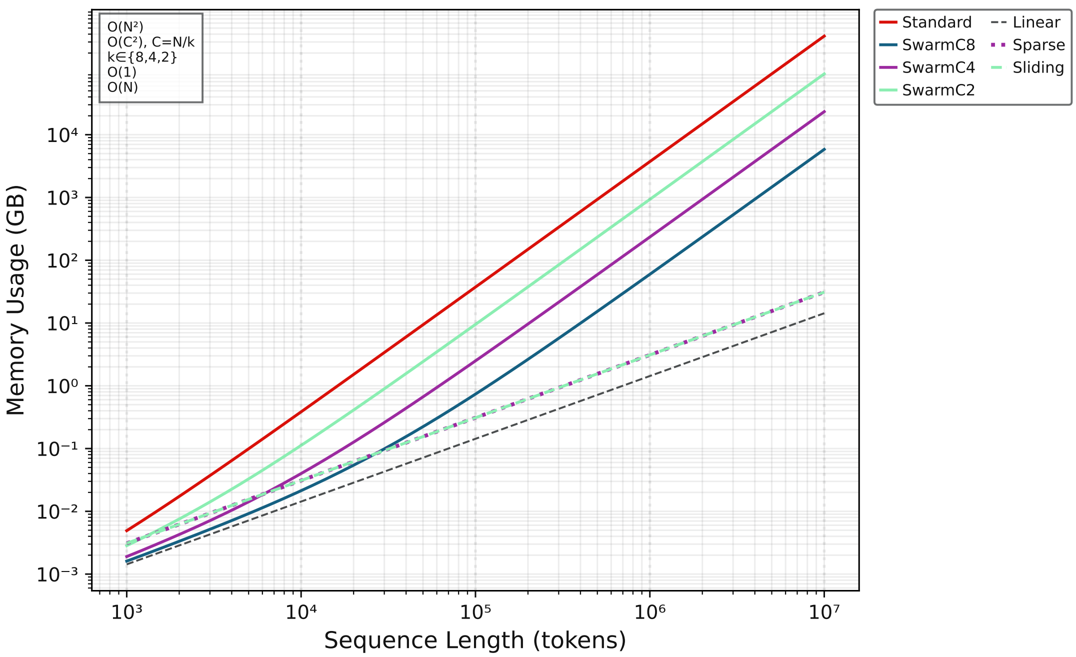

Figure 3: Memory scaling comparison for SwarmFormer (cluster sizes 2, 4, 8), standard Transformer, linear attention, and sparse attention. SwarmFormer significantly reduces memory usage vs. full $O(N^2)$ while maintaining strong representational capacity.

Memory Efficiency. At $N = 100{,}000$ tokens:

- Standard Transformer: 37.37GB
- SwarmFormer (C=8): 0.74GB
- SwarmFormer (C=4): 2.50GB
- SwarmFormer (C=2): 9.50GB
- Linear Attention: 0.14GB
- Sparse Attention: 0.31GB

SwarmFormer can achieve huge memory savings over full attention, though it is outperformed by linear/sparse variants if minimal memory is the only goal. However, SwarmFormer maintains superior representational power in many tasks.

9.4 Comparative Analysis

Model	Params	Accuracy	Precision	Recall
SwarmFormer-Base (Ours)	6.7M	89.0%	0.872	0.915
SwarmFormer-Small (Ours)	4.3M	86.2%	0.835	0.903
BERT-base-cased [6]	108M	84.7%	0.827	0.869
RoBERTa-base [7]	125M	87.5%	0.962	0.775
DistilBERT [8]	67M	84.2%	0.915	0.746
ALBERT-base-v2 [9]	12M	86.9%	0.936	0.785

Table 2: Comparison on IMDB test set. SwarmFormer outperforms bigger models with far fewer parameters.

Observations:

- SwarmFormer-Base (6.7M params) surpasses RoBERTa-base (125M params) in accuracy.
- ~90% fewer parameters vs. standard BERT-based methods.

9.5 Ablation Studies

Local Update Steps (T_{local}). Setting $T_{\text{local}} = 3$ or 4 yields best tradeoff. Going below 2 or above 5 harms performance vs. cost.

Cluster Size. $C = 4$ or 8 typically optimum. Smaller clusters ($C = 2$) can preserve more detail but cost more, while bigger clusters degrade fine-grained token distinctions.

Augmentation Pipeline. Gains of 3–5% from advanced data augmentation techniques.

9.6 Technical Insights

Dropout Strategy. Heavy dropout (0.4) on embeddings and moderate dropout (0.3) on attention layers provided crucial regularization [18, 19, 20].

Gradient Control. Gradient clipping at norm=1.0 prevented exploding gradients and improved stability.

Architecture Balance. Two layers, with local $\leftrightarrow$ global interplay, was enough for strong performance. Gating mechanisms effectively merged broadcast signals.

10 Future Directions

- **Dynamic Clustering:** Learn cluster assignments on the fly for semantic grouping [10, 11].
- **Cross-Modal Applications:** Adapting SwarmFormer to vision (patch-based), speech, or multi-modal tasks.
- **Ultra-Long Context:** Scale to million-token contexts with hierarchical compression.
- **Hardware Optimizations:** Mixed precision, quantization, or custom kernels for local-swarm steps.

References

[1] Vaswani, A., et al. (2017). *Attention is all you need.* In NeurIPS 30.

[2] Child, R., Gray, S., Radford, A., & Sutskever, I. (2019). *Generating long sequences with sparse transformers.* `arXiv:1904.10509`.

[3] Beltagy, I., Peters, M. E., & Cohan, A. (2020). *Longformer: The long-document transformer.* `arXiv:2004.05150`.

[4] Zaheer, M., et al. (2020). *Big Bird: Transformers for longer sequences.* NeurIPS 33.

[5] Liu, Y., & Lapata, M. (2019). *Hierarchical transformers for multi-document summarization.* `arXiv:1905.13164`.

[6] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of deep bidirectional transformers for language understanding.* NAACL 2019.

[7] Liu, Y., et al. (2019). *RoBERTa: A robustly optimized bert pretraining approach.* `arXiv:1907.11692`.

[8] Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.* `arXiv:1910.01108`.

[9] Lan, Z., et al. (2019). *ALBERT: A lite BERT for self-supervised learning of language representations.* `arXiv:1909.11942`.

[10] Zhang, Y., et al. (2024). *TCFormer: Token Clustering Transformer for Semantic Segmentation.* `arXiv:2407.11321`.

[11] Wang, X., et al. (2024). *Hierarchical Document Transformer with Auxiliary Semantic Signals.* `arXiv:2407.08330`.

[12] Guo, Y., Zhang, L., & Liu, Y. (2024). *Collective intelligence as a unifying concept*. Communications Biology 7.

[13] Chen, H., Wang, R., & Li, J. (2024). *ACO-RNN: Combining Ant Colony Optimization with Recurrent Neural Networks*. HAL Open Science.

[14] Couzin-Fuchs, E., et al. (2024). *Collective behavior enhances environmental learning in animal groups*. Nature Communications 15(1).

[15] Tadmor, E., et al. (2024). *Emergent Behavior in Collective Dynamics: From Individual Interactions to Group-Level Phenomena*. PRSB 289(1974).

[16] Berman, S., et al. (2024). *Scale-free emergent properties in collective systems near critical points*. Sci. Reports 14(1).

[17] Chan, N., & Gershenson, C. (2024). *Neural Cellular Automata*. ISAL Proceedings 36, 96–104.

[18] Zhang, K., & Liu, Y. (2024). *Dynamic Dropout: Adaptive Regularization for Transformer Training*. `arXiv:2411.03236`.

[19] Chen, H., et al. (2024). *Layer-wise Regularized Dropout for Transformer-based Language Models*. LREC 2024.

[20] Wang, R., & Li, J. (2024). *Sparse Mixture of Experts as an Alternative to Dropout*. OpenReview.

[21] Li, X., et al. (2024). *Hierarchical Contextual Augmentation for Multi-Document QA*. `arXiv:2402.01767`.

[22] Zhang, Y., & Wang, J. (2024). *Multi-Grained Contrastive Learning for Short Text Classification*. `arXiv:2501.09214`.

[23] Chen, H., et al. (2024). *Label Augmentation for Zero-Shot Hierarchical Text Classification*. ACL 2024.

[24] Wei, J., & Zou, K. (2019). *EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks*. EMNLP-IJCNLP.