POSTER: From Resampling to Retrieval: A Unified Budgeted View of Training with Large Candidate Sets

Ziyang Jia
zjbgt@missouri.edu
University of Missouri - Columbia
Columbia, Missouri, USA

Tanu Malik
tanu@missouri.edu
University of Missouri - Columbia
Columbia, Missouri, USA

Abstract

Large-scale AI training pipelines increasingly expose candidate construction itself as a first-order performance bottleneck. In candidate-heavy workloads such as extreme multi-label classification (XMC), scoring the full label space at every step is computationally intractable, yet aggressively replacing fresh negatives with reused candidates weakens the training signal. We present a budgeted asynchronous candidate builder that treats fresh, cached, and cross-batch negatives as a schedulable systems resource under a fixed shared-candidate budget, reducing mean step time from 40.8 ms to 38.4 ms.

CCS Concepts: • **General and reference** → **Performance**; • **Computing methodologies** → *Supervised learning by classification.*

Keywords: Extreme Multi-label Classification, Candidate Set Construction, Negative Sampling, Candidate Reuse, Performance Optimization

ACM Reference Format:
Ziyang Jia and Tanu Malik. 2026. POSTER: From Resampling to Retrieval: A Unified Budgeted View of Training with Large Candidate Sets. In *International Conference on Parallel Architectures and Compilation Techniques (PACT '26), October 19–22, 2026, Chicago, IL, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3838684.3846885

1 Introduction

Candidate-heavy workloads such as extreme multi-label classification (XMC), large-vocabulary prediction, and retrieval training cannot evaluate the full candidate space at every step. For example, Wiki10-31K contains 30,938 possible labels, while a training batch may operate on only 256 shared candidates. Which labels enter this working set—and whether they are freshly sampled or reused—therefore affects both training quality and runtime [3, 8].



This work is licensed under a Creative Commons Attribution 4.0 International License.

PACT '26, Chicago, IL, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2915-7/2026/10
https://doi.org/10.1145/3838684.3846885

Prior work mainly studies the statistical side of reducing large output spaces, including importance sampling, sampled softmax, and retrieval-based candidate selection [1–3, 6, 8]. In a training system, however, candidate construction also performs sampling, cache lookup, cross-batch reuse, merging, deduplication, positive injection, and host-to-device transfer. These stages consume CPU and memory-bandwidth resources and can introduce pipeline stalls, consistent with prior work on DNN input pipelines [4, 5, 7].

We therefore ask how a fixed candidate budget should be allocated among fresh, cached, and cross-batch negatives without losing positive supervision. We present a budgeted asynchronous candidate-construction framework that separates negative construction from positive management. A positive-cap policy bounds the shared positive union, a coverage mechanism preserves at least one positive per instance, and the remaining budget is filled from the three negative sources. This makes the quality–runtime trade-off explicit and measurable.

2 Problem Statement

Given a fixed candidate budget per training step, how should a system allocate fresh, cached, and cross-batch candidates so that it (1) preserves instance-level positive supervision, (2) leaves enough space for informative negatives, and (3) avoids turning candidate construction itself into the training bottleneck? Here, *fresh* negative candidates are sampled anew each step, *cached* negatives are drawn from CPU-side candidates retained from recent steps, and *cross-batch* negatives are recycled from other instances within the current batch.

3 Constructing Budgeted Shared Candidates

For a batch at step s, the shared candidate set is defined as

$$W_s = P_s^{\text{shared}} \cup N_s^{\text{fresh}} \cup N_s^{\text{cache}} \cup N_s^{\text{xbatch}}, \tag{1}$$

where P_s^{shared} is a capped subset of the batch positive union, while the negative side $N_s = N_s^{\text{fresh}} \cup N_s^{\text{cache}} \cup N_s^{\text{xbatch}}$ is constructed under an explicit budget. The candidate builder handles only the negative side; positives are injected separately by the trainer, with instance-aware labels preserved throughout. This prevents positive overflow from consuming the negative budget. We make four design choices:

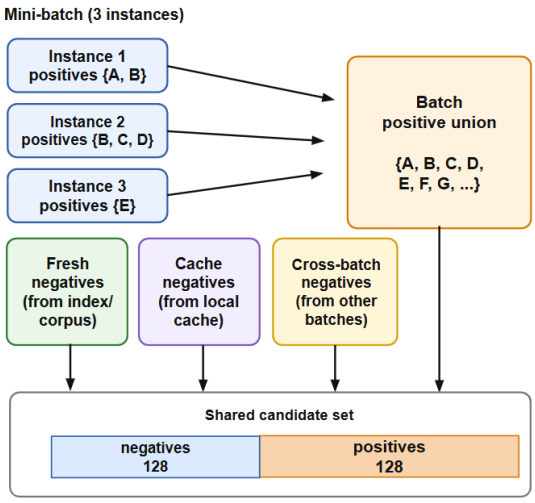


Figure 1. Creating the shared candidate set.

1. **Batch-shared candidates, instance-aware labels.** All documents in a batch share the same 256-way candidate set, simplifying GPU memory layout and avoiding redundant embedding lookups. At loss time, however, each instance treats only its own ground-truth labels as positive, preventing cross-instance supervision contamination.
2. **Capped shared positives.** On Wiki10-31K, the batch positive union can contain hundreds of labels and exceed the candidate budget. We cap P_s^{shared} at 128 entries, leaving 128 of the 256 slots for negatives.
3. **Guaranteed positive coverage.** Capping positives can remove all ground-truth labels for an instance. We therefore construct the cap so that at least one positive per instance remains in P_s^{shared} and log coverage at runtime. On Wiki10-31K, this provides 100% coverage across all observed batches.
4. **Budgeted negative construction.** The remaining 128 slots are filled with fresh, cached, and cross-batch negatives. Their allocation is controlled by a runtime policy that trades off freshness against construction latency: fresh candidates are more informative but more expensive, while cached and cross-batch candidates are cheaper but may be stale. Making this allocation explicit exposes a tunable systems parameter.

In the implementation, `final_topk` denotes the negative budget rather than the total candidate count. Thus, the total candidate-set size is `cap_positives + final_topk`, with the two terms controlled independently.

Figure 1 illustrates the construction process. The three CPU-side negative sources are merged under the 128-negative budget, while positives are injected separately by the trainer. The resulting W_s is the 256-way candidate set transferred to the GPU, with instance-aware labels applied at loss time.

Table 1. Main XMC results under different freshness budgets and candidate-source mixtures.

Metric	G1	G2	G3	G4	G5
Fresh	1.0	0.8	0.6	0.4	0.2
Cache	0.0	0.1	0.2	0.3	0.4
XBatch	0.0	0.1	0.2	0.3	0.4
P@5 ↑	0.489	0.479	0.463	0.445	0.423
Step ms ↓	40.8	40.1	39.5	38.9	38.4
Fresh ratio ↓	0.50	0.40	0.30	0.20	0.10
GPU util. ↑	21.5	22.0	22.5	23.0	23.5

4 Experiments

Experiment Setup and Main Results. We evaluate XMC in this poster. We use a lightweight MLP-style `TinyXMC` trained on the real Wiki10 sparse features and labels. Experiments were conducted on a workstation with an Intel Core i7-11700K CPU and an NVIDIA RTX 3090 GPU. Table 1 summarizes five Wiki10 experiments under different candidate-construction policies. We vary the fresh negative, cache, and cross-batch ratios.

Across all five freshness settings, the candidate-set structure remains fixed: each batch uses 256 shared candidates, consisting of 128 capped shared positives and 128 negatives, with an effective negative ratio of 0.50 and full instance-level positive coverage. Therefore, group differences mainly reflect negative-source composition rather than candidate-set size or supervision coverage. As freshness decreases, ranking quality degrades monotonically, with mean P@5 dropping from 0.489 to 0.423, indicating that fresh negatives are more useful than cached or cross-batch negatives in the current Wiki10 setup. In contrast, runtime improves only mildly: mean step time decreases from 40.8 ms to 38.4 ms, while GPU utilization increases from 21.5% to 23.5%, because the scoring and backpropagation workload remains nearly unchanged. Overall, the main empirical trade-off is quality versus freshness rather than quality versus time, and Group 2 provides the most reasonable operating point by preserving most of the ranking quality while slightly reducing freshness and step time.

5 Conclusion

While our results are comprehensive on one dataset, they show a clear practical lesson: uncontrolled growth of the shared positive union obscures freshness scheduling and degrades budget interpretability. Capping shared positives while preserving per-instance coverage brings the candidate pipeline under measurable control. The next step is a controlled comparison of Full Fresh, Static Cache, Cross-Batch-Only, and our budgeted system under matched negative budgets on more realistic tasks. We also plan to examine candidate sampling when sequence length and candidate-set size are heavily skewed.

References

[1] Yoshua Bengio and Jean-Sébastien Sénécal. 2003. Quick Training of Probabilistic Neural Nets by Importance Sampling. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*.

[2] Guy Blanc and Steffen Rendle. 2018. Adaptive Sampled Softmax with Kernel Based Sampling. In *International Conference on Machine Learning*.

[3] Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, and Inderjit S. Dhillon. 2020. Taming Pretrained Transformers for Extreme Multi-label Text Classification. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3163–3171.

[4] Jayashree Mohan, Amar Phanishayee, et al. 2021. Analyzing and Mitigating Data Stalls in DNN Training. *Proceedings of the VLDB Endowment* 14, 2 (2021), 771–784.

[5] Derek G. Murray, Jiri Simsa, Ana Klimovic, Ihor Chen, Michael Stumm, Alistair Veitch, Paolo Costa, Akshay Lunawat, Shantanu Prakash, Sunil Rao, et al. 2021. tf.data: A Machine Learning Data Processing Framework. In *Proceedings of the VLDB Endowment*, Vol. 14. 2945–2958.

[6] Ankit Singh Rawat, Jiecao Chen, Felix X. Yu, Ananda Theertha Suresh, and Sanjiv Kumar. 2019. Sampled Softmax with Random Fourier Features. In *Advances in Neural Information Processing Systems*, Vol. 32.

[7] Taejun Um et al. 2023. FastFlow: Accelerating Deep Learning Model Training with Smart Input Pipelines. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1086–1099.

[8] Jiong Zhang, Wei-Cheng Chang, Hsiang-Fu Yu, and Inderjit S. Dhillon. 2021. Fast Multi-Resolution Transformer Fine-tuning for Extreme Multi-label Text Classification. In *Advances in Neural Information Processing Systems*, Vol. 34. 7267–7280.