Fast Listwise Passage Reranking via Semantic Tokens

Omkar Prabhune[1]*, **Xijun Wang**[2], **Nan Bi**[2], **Haiyan Wang**[2],
Shima Alizadeh[2], **Anoop Jain**[2], **Yu Lou**[2], **Junbang Liang**[2]
[1]Purdue University [2]Amazon
oprabhun@purdue.edu
{xijunw, nanbi, haiyanww, alizshim, anoopjai, ylou, junbanl}@amazon.com

Abstract

Listwise passage reranking often achieves strong ranking performance, but it incurs substantial computational cost. We propose FLASH, an efficient listwise reranking framework based on semantic token compression and single-step listwise scoring. FLASH represents each query–passage pair with a small set of query-aware semantic tokens and jointly scores all candidates in a single decoder step, enabling query-conditioned joint scoring without expensive token-level listwise processing. We further introduce multi-cardinality training, which allows a single model to support different semantic-token budgets at inference time. We also propose FLASH-LITE, a lightweight variant with offline passage encoding for low-cost online inference. Across TREC and BEIR benchmarks, FLASH improves average ranking effectiveness at comparable online latency and slightly lower FLOPs, while FLASH-Lite substantially reduces query-time cost with a modest effectiveness trade-off. FLASH-LITE enables offline passage encoding, substantially reducing online query-time cost while maintaining competitive reranking performance. Code is provided in the supplementary material.

1 Introduction

Passage reranking is a central component of modern information retrieval systems, where an efficient retriever first selects a small set of candidate passages from a large corpus, and a reranker then reorders these candidates according to their estimated relevance to the query. Language models improve reranking effectiveness, but their high computational cost limits large-scale and latency-sensitive deployment.

Existing reranking methods can be broadly categorized into pointwise and listwise approaches. Pointwise methods (Nogueira and Cho, 2019; Devlin et al., 2019; Nogueira et al., 2020; Zhuang et al., 2023) independently score each query–passage pair, offering strong efficiency and scalability but fail to capture interactions across candidate passages. In contrast, listwise methods (Sun et al., 2023; Pradeep et al., 2023, 2024; Yoon et al., 2024; Tamber et al., 2023) jointly reason over multiple passages and generally achieve stronger ranking performance. However, most listwise rerankers operate directly over the full text of the query and candidate passages. This design enables rich cross-passage comparison, but it also creates long input sequences and often requires autoregressive generation or multiple inference passes, resulting in high latency and poor scalability.

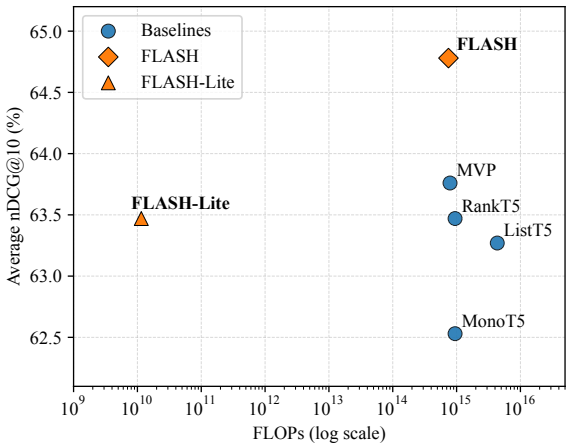


Figure 1: Effectiveness–efficiency trade-off under the same T5-base backbone. FLASH improves ranking quality while reducing FLOPs, and FLASH-LITE provides substantially lower online query-time cost with competitive reranking performance.

Recent efficient listwise rerankers (Reddy et al., 2024; Liu et al., 2024; Na et al., 2025) attempt to reduce this high cost of listwise reasoning through single-step decoding or using latent passage representations. However, existing approaches either still rely on expensive token-level processing (Reddy et al., 2024), suffer from information loss due to aggressive compression (Liu et al.,

*Work done during an internship at Amazon.

Proceedings of the 15th International Joint Conference on Natural Language Processing and the 5th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, pages 4897–4910
November 6-10, 2026 ©2026 Association for Computational Linguistics

2024), or lack effective joint reasoning across multiple passage representations (Na et al., 2025).

To address these limitations, we propose FLASH, an efficient listwise reranker that compresses each query–passage pair into a small set of query-aware semantic tokens and jointly scores all candidates in a single decoder step. We further introduce a cardinality-configurable objective, allowing the same model to operate with different numbers of semantic tokens at inference time, and propose FLASH-LITE, which decouples query and passage encoding so passage representations can be precomputed offline.

Our model combines several desirable properties: (i) efficient query-aware compression of candidate passages, (ii) expressive multi-vector representations (semantic tokens), (iii) single-step listwise scoring jointly over semantic tokens, (iv) support for offline passage encoding, and (v) configurable inference budgets within a single model. To the best of our knowledge, no prior method simultaneously achieves all these properties. We summarize our contributions as follows:

- **FLASH**, an efficient listwise reranking framework built on a **semantic token compressor** and **query-conditioned joint scoring** that supports **cardinality-configurable inference** within a single model. The compressor transforms the passages into compact semantic tokens, while the one-shot decoder jointly scores all passages with query-conditioned joint scoring.
- **FLASH-LITE**, a lightweight variant that supports **offline passage encoding** and is trained with both score-level and semantic-token-level distillation, combining pointwise efficiency with listwise effectiveness.
- **Extensive Evaluation.** Experiments on in- and out-of-domain benchmarks show that FLASH improves ranking effectiveness while maintaining comparable online inference cost to strong efficient reranking baselines. FLASH-LITE targets a complementary, efficiency-oriented operating point.

2 Related Work

Neural passage reranking refines candidates from sparse lexical retrievers such as BM25 (Robertson and Zaragoza, 2009) or dense retrievers (Karpukhin et al., 2020; Xiong et al., 2021). Existing methods can be broadly categorized into pointwise and listwise reranking.

Pointwise Reranking. Pointwise methods independently score each query–passage pair (Nogueira and Cho, 2019; Devlin et al., 2019). Sequence-to-sequence variants such as monoT5 (Nogueira et al., 2020) and RankT5 (Zhuang et al., 2023) further improve effectiveness. These models are efficient and scalable but fail to capture inter-passage interactions.

Listwise Reranking. Listwise methods jointly model multiple passages, enabling cross-passage reasoning. Token-level approaches such as RankGPT (Sun et al., 2023), RankVicuna (Pradeep et al., 2023), RankZephyr (Pradeep et al., 2024), ListT5 (Yoon et al., 2024), and LiT5 (Tamber et al., 2023) operate on raw tokens but incur high cost due to autoregressive or multi-pass decoding.

Efficient Listwise Reranking. Recent work improves listwise efficiency using single-step decoding or compact passage representations. PE-Rank (Liu et al., 2024) supports offline passage encoding by using a single passage representation, but suffers with information loss due to aggressive compression and still uses generative decoding. MVP (Na et al., 2025) introduces multi-vector representations with non-generative scoring, but scores them independently and aggregates the results, limiting joint reasoning across representations. Moreover, constructing these representations through the encoder introduces significant computational overhead due to quadratic self-attention complexity. These methods also use fixed representation budgets, unlike our cardinality-configurable inference.

As summarized in Table 1, FLASH differs from prior efficient listwise rerankers by combining compact multi-vector passage representations with joint single-step listwise scoring. Compared to PE-Rank, it avoids compressing each passage into a single vector; compared to MVP, it jointly reasons over multiple semantic tokens instead of scoring them independently. FLASH-LITE further supports offline passage encoding, while both variants allow configurable semantic-token cardinality at inference time.

3 Proposed Methodology

Given a query q and a candidate set $\mathcal{P} = \{p_i\}_{i=1}^{N}$ retrieved by a first-stage retriever, a reranker learns a scoring function $\mathbf{s} = f_\theta(q, \mathcal{P}) \in \mathbb{R}^N$, where $\mathbf{s} = [s_1, \ldots, s_N]$ assigns a relevance score s_i to

Method	Listwise	# Passage Vectors Used per Decoding Step	One-Step Decoding	Offline Passage Encoding	Configurable Cardinality
MonoT5 / RankT5	×	L_p	✓	×	×
ListT5	✓	L_p	×	×	×
RankZephyr / RankVicuna	✓	L_p	×	×	×
PE-Rank	✓	1	×	✓	×
MVP	✓	1	✓	×	×
FLASH (Ours)	✓	$m \ll L_p$	✓	×	✓
FLASH-LITE (Ours)	✓	$m \ll L_p$	✓	✓	✓

Table 1: Comparison of reranking methods. L_p is passage length and $m \ll L_p$ is the number of summary vectors.

each passage p_i. The final ranking is obtained by sorting passages according to their scores.

In the following sections, we first describe the FLASH architecture and its components, followed by FLASH-LITE, and finally the training objectives used for optimization.

3.1 FLASH Architecture

FLASH avoids expensive token-level listwise processing by encoding each query–passage pair independently and compressing the resulting query-aware representation into a small set of semantic tokens. It then jointly scores the semantic tokens from all candidate passages using a single-step decoder, enabling global comparison over compact representations. As shown in Figure 2, FLASH consists of input construction and pointwise encoding, semantic token compression, and cardinality-configurable listwise scoring.

3.1.1 Input Construction and Encoding

Given a query q and a set of N candidate passages $\mathcal{P} = \{p_i\}_{i=1}^{N}$, we construct the query input $x_q = \texttt{Query: } q$ and each query–passage input $x_i = \texttt{Query: } q \texttt{ | Context: } p_i$. Pairing each passage with the query enables early query–passage interaction within the encoder, allowing the resulting semantic tokens to capture query-aware relevance information.

Each input sequence x_i is encoded independently: $\mathbf{H}_i = \text{Encoder}(x_i) \in \mathbb{R}^{L_i \times d}$, where L_i is the length of sequence x_i and d is the hidden dimension. We apply the same encoder to the query sequence x_q to obtain $\mathbf{H}_q \in \mathbb{R}^{L_q \times d}$, where L_q is the sequence length of the query. Since each query–passage pair is encoded independently, this stage avoids constructing a single long input over all candidate passages. At the same time, concatenating the query with each passage within x_i enables fine-grained query–passage interaction before semantic token compression.

3.1.2 Semantic Token Compression

To reduce the cost of listwise scoring, FLASH compresses each encoded query–passage sequence into a small set of semantic tokens. Given the encoder output $\mathbf{H}_i \in \mathbb{R}^{L_i \times d}$, a lightweight Semantic Token Compressor (STC) produces K semantic tokens, $\mathbf{V}_i = \text{STC}(\mathbf{H}_i) \in \mathbb{R}^{K \times d}$, where $K \ll L_i$.

The STC consists of K learnable compressor vectors, where each compressor vector attends over the encoded token representations and produces one semantic token through attention-based weighted aggregation:

$$\begin{aligned} \mathbf{v}_{i,k} &= \sum_{t=1}^{L_i} \alpha_{i,k,t} \mathbf{h}_{i,t}, \\ \alpha_{i,k,t} &= \frac{\exp(\mathbf{c}_k^\top \mathbf{h}_{i,t}/\sqrt{d})}{\sum_{t'=1}^{L_i} \exp(\mathbf{c}_k^\top \mathbf{h}_{i,t'}/\sqrt{d})}, \end{aligned} \tag{1}$$

where $\mathbf{h}_{i,t}$ is the t-th token in $\mathbf{H}_i$, and $\mathbf{c}_k$ is the k-th learnable compressor vector.

We apply the same STC operation to the encoded query representation $\mathbf{H}_q$ to obtain semantic tokens $\mathbf{V}_q \in \mathbb{R}^{K \times d}$ for the query. Unlike MVP (Na et al., 2025) that constructs multiple passage representations through additional encoder computation, STC extracts semantic tokens directly from the encoder outputs using a lightweight attention-based aggregation step, making it more efficient by avoiding additional encoder self-attention, while also allowing each semantic token to directly aggregate information from the full query-aware token sequence.

3.1.3 Listwise Scoring

Although the previous stages produce query-aware semantic tokens for each passage, they process candidates independently and therefore do not yet model interactions across the candidate set. The goal of the final scoring stage is to convert these pointwise compressed representations into a listwise ranking decision. The final scoring stage consists of the following steps.

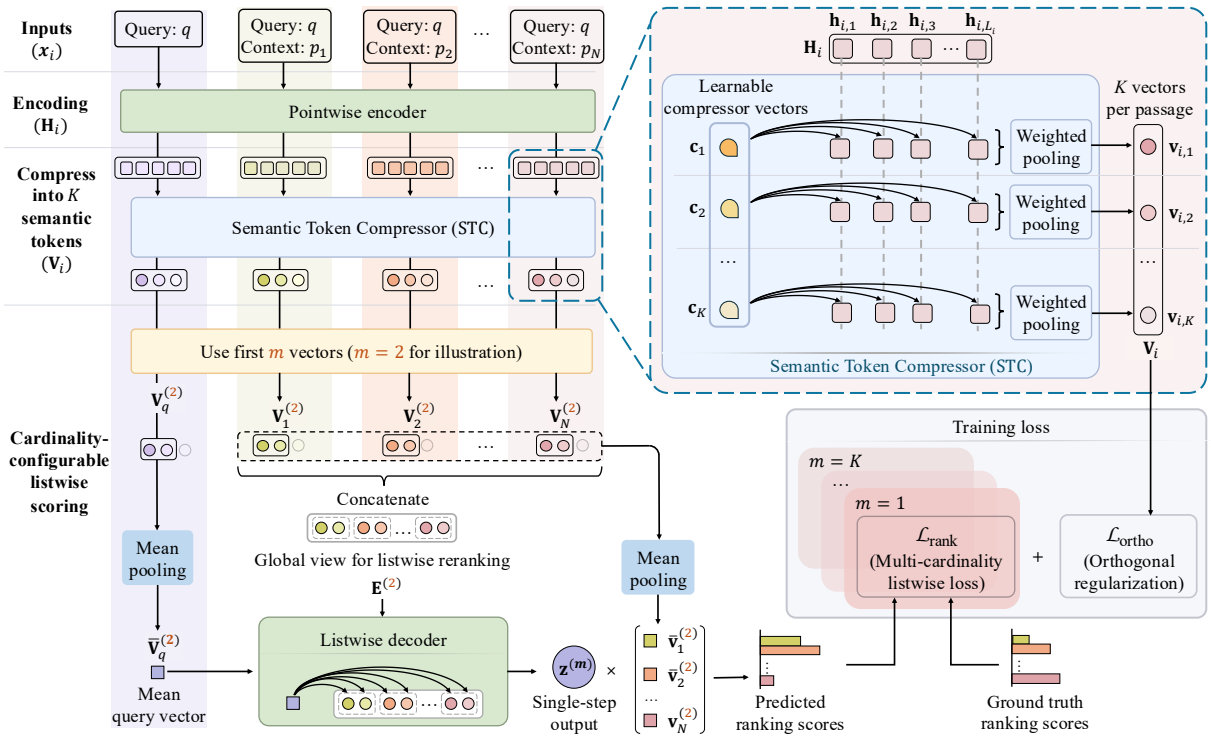


Figure 2: Overview of FLASH. The framework compresses query-aware passage representations into compact semantic-token representations and performs single-step listwise scoring.

Cardinality Selection. To enable a single trained model to support multiple semantic-token budgets at inference time, FLASH is trained to operate with different cardinalities $m \leq K$.

For a selected cardinality m, we define $\mathbf{V}_i^{(m)} = [\mathbf{v}_{i,1}; \ldots; \mathbf{v}_{i,m}] \in \mathbb{R}^{m \times d}$ and $\mathbf{V}_q^{(m)} = [\mathbf{v}_{q,1}; \ldots; \mathbf{v}_{q,m}] \in \mathbb{R}^{m \times d}$. As described later in Section **??**, we train the model across all cardinalities $m = 1, \ldots, K$. Since each cardinality uses a prefix of the semantic tokens, token k participates in the ranking objective for all $m \geq k$. Thus, earlier tokens receive supervision across more cardinality settings, inducing an ordered prefix representation.

For a given cardinality m, the scoring stage uses only the first m semantic tokens of each passage. This explicitly optimizes the model to use the first m tokens effectively for ranking at each supported cardinality, rather than relying only on the full set of K tokens. Consequently, the same trained model can operate under different semantic-token budgets at inference time without retraining. We empirically validate the learned token ordering through an exhaustive subset analysis in Appendix B.

Listwise Memory Construction. To enable listwise interaction across candidates, we concatenate the selected semantic tokens from all passages:

$$\mathbf{E}^{(m)} = [\mathbf{V}_1^{(m)}; \ldots; \mathbf{V}_N^{(m)}] \in \mathbb{R}^{(N \cdot m) \times d}. \quad (2)$$

The resulting memory contains semantic-token representations from all candidate passages simultaneously. Since this memory contains only $N \cdot m$ vectors, FLASH performs listwise reasoning over compact semantic token representations rather than full token-level passage sequences.

Single-Step Decoding. To perform decoding in a single step, the decoder requires a single input representation. While standard encoder–decoder models typically start decoding from a special decoder-start token, FLASH instead initializes the decoder with a query-dependent representation. Specifically, we aggregate the selected query semantic tokens as $\bar{\mathbf{v}}_q^{(m)} = \frac{1}{m} \sum_{k=1}^{m} \mathbf{v}_{q,k}$. Using $\bar{\mathbf{v}}_q^{(m)}$ as the decoder input conditions the decoding process on the query while allowing the decoder to attend to all candidate passages through listwise memory.

The decoder then attends to the listwise memory in a single step as $\mathbf{z}^{(m)} = \text{Decoder}(\bar{\mathbf{v}}_q^{(m)}, \mathbf{E}^{(m)})$. Since $\mathbf{E}^{(m)}$ contains semantic tokens from all candidate passages, the decoder output $\mathbf{z}^{(m)}$ is candidate-set dependent and captures query-conditioned information after jointly attending to all candidates. Consequently, each passage score is candidate-set dependent through $\mathbf{z}^{(m)}$: changing one candidate can affect the scores of other candidates even when their semantic representa-

tions remain unchanged. We verify this behavior using a controlled candidate-replacement analysis in Appendix C.1.

Passage Scoring. After single-step decoding, $\mathbf{z}^{(m)}$ serves as a query-conditioned listwise representation that has attended to the semantic tokens of all candidate passages. To compute one relevance score for each passage, we derive a passage-level representation by averaging its selected semantic tokens as $\bar{\mathbf{v}}_i^{(m)} = \frac{1}{m}\sum_{k=1}^{m} \mathbf{v}_{i,k}$. The relevance score for passage p_i is computed by matching this passage representation with the query-conditioned listwise decoder output, $s_i^{(m)} = \left\langle \mathbf{z}^{(m)}, \bar{\mathbf{v}}_i^{(m)} \right\rangle$.

This design keeps both decoding and scoring in a single-step formulation: the decoder produces one query-conditioned listwise representation, and all candidate passages are scored using dot products against their aggregated semantic-token representations. Unlike approaches that score each representation independently and aggregate the resulting scores (Na et al., 2025), FLASH performs unified listwise scoring after jointly attending to semantic tokens from all candidate passages.

3.2 FLASH-LITE

FLASH-LITE further improves efficiency by decoupling passage encoding from the query, enabling passage representations to be computed offline. A detailed illustration is provided in Appendix A (Figure 3). We describe the main steps below.

Input Preparation. We construct $N+1$ input sequences, with $x_q = \texttt{Query: } q$ for the query and $x_i = \texttt{Context: } p_i$ for each passage $i = 1, \ldots, N$. Unlike FLASH, the query is not concatenated with passages. This separation allows passage representations to be computed once and reused across queries, enabling efficient large-scale deployment. The offline passage-encoding cost is therefore shifted outside the online computation cost.

Pointwise Encoding. All sequences are encoded independently as $\mathbf{H}_q = \mathrm{Encoder}(x_q) \in \mathbb{R}^{L_q \times d}$ and $\mathbf{H}_i = \mathrm{Encoder}(x_i) \in \mathbb{R}^{L_i \times d}$. Decoupled encoding allows the final-layer token-level passage encoder output $\mathbf{H}_i$ to be precomputed offline and cached. At inference time, the query is encoded online, while the cached passage representations are reused and fused with the query through lightweight online layers. Query encoding is included in the reported online latency and FLOPs. Caching $\mathbf{H}_i \in \mathbb{R}^{L_i \times d}$ requires $L_i d$ values per passage. For T5-base ($d = 768$) with FP16 storage, this corresponds to $1536L_i$ bytes, or approximately 192 KB per passage under our experimental sequence length. The cached passage states are query-agnostic, while the final semantic tokens are query-conditioned through online fusion.

Lightweight Fusion. To restore query–passage interaction after independent encoding, without re-encoding each passage from scratch, we perform a lightweight fusion step. For each passage, we concatenate the encoded query representation with the precomputed passage representation along the sequence dimension, $\mathbf{C}_i^{(0)} = [\mathbf{H}_q; \mathbf{H}_i] \in \mathbb{R}^{(L_q+L_i)\times d}$.

We then apply L_f self-attention layers:

$$\mathbf{C}_i^{(\ell)} = \mathrm{LN}\Big(\mathbf{C}_i^{(\ell-1)} + \mathrm{SelfAttn}(\mathbf{C}_i^{(\ell-1)})\Big), \quad \ell = 1, \ldots, L_f. \tag{3}$$

where LN denotes layer normalization, SelfAttn denotes a self-attention layer, and L_f is the number of lightweight fusion layers. This fusion stage enables query and passage tokens to interact and partially recovers query-aware interactions. Importantly, it is significantly cheaper than re-running the full encoder for each query–passage pair, providing a favorable trade-off between efficiency and expressiveness.

From this point onward, we use the fused representation $\mathbf{C}_i^{(L_f)}$ in place of $\mathbf{H}_i$ for semantic token compression, and the subsequent cardinality-configurable listwise scoring is identical to FLASH.

3.3 Training

Multi-Cardinality Listwise Loss. Both FLASH and FLASH-LITE are trained using a multi-cardinality objective that supervises ranking performance across all semantic-token cardinalities $m = 1, \ldots, K$. Training across multiple values of m ensures that a single model can operate effectively under different inference budgets. This enables configurable cardinality at inference time without retraining. We construct reciprocal relevance scores $y_i = 1/\mathrm{rank}(i)$ and convert them into a target distribution $\tilde{\mathbf{y}} = \mathrm{softmax}(\mathbf{y}/\tau)$, where $\mathbf{y} = [y_1, \ldots, y_N]$, $\mathrm{rank}(i)$ denotes the supervision rank of passage p_i, and τ is a tempera-

ture parameter. For each m, the model produces scores $\mathbf{s}^{(m)}$, which are converted into $\mathbf{p}^{(m)} = \mathrm{softmax}(\mathbf{s}^{(m)}/\tau)$. The loss is:

$$\mathcal{L}_{\text{rank}} = -\sum_{m=1}^{K} w_m \sum_{i=1}^{N} \tilde{y}_i \log p_i^{(m)}, \quad (4)$$

where w_m is the corresponding weight. We apply equal weight $w_m = 1/K$ to all loss components. This encourages the model to concentrate the most informative semantic tokens in the leading positions, so that using a smaller m at inference time results in only a limited degradation in performance.

Orthogonal Regularization. To encourage diversity among the K semantic tokens per passage and prevent mode collapse, we apply:

$$\mathcal{L}_{\text{orth}} = \frac{1}{B \cdot N} \sum_{b,i} \left\| \hat{\mathbf{V}}_{b,i} \hat{\mathbf{V}}_{b,i}^{\top} - \mathbf{I} \right\|_F^2, \quad (5)$$

where B is the batch size, $\hat{\mathbf{V}}_{b,i}$ is the row-wise ℓ_2-normalized version of the K semantic tokens for passage i in batch item b, $\mathbf{I} \in \mathbb{R}^{K \times K}$ is the identity matrix, and $\| \cdot \|_F$ denotes the Frobenius norm. Without this regularization, multiple semantic tokens may collapse to similar representations, reducing the benefit of multi-vector modeling.

The total loss is:

$$\mathcal{L}_{\text{FLASH}} = \mathcal{L}_{\text{rank}} + \lambda_{\text{orth}} \mathcal{L}_{\text{orth}}, \quad (6)$$

where λ_{orth} is the weight given to the regularization component.

Knowledge Distillation. To improve the ranking performance of FLASH-LITE, we distill knowledge from a pretrained FLASH model at both the ranking level and the representation level. Since FLASH-LITE employs a more efficient but weaker interaction mechanism, distillation helps recover performance by transferring richer ranking signals and multi-vector representations from the teacher.

Logit-level distillation. For consistency with multi-cardinality training, we apply logit-level distillation at each cardinality m. The distillation loss on the final ranking logits is:

$$\mathcal{L}_{\text{KD}}^{\text{logit}} = \tau_{\text{KD}}^2 \sum_{m=1}^{K} w_m \mathrm{KL}\left(\mathbf{p}_t^{(m)} \| \mathbf{p}_s^{(m)} \right), \quad (7)$$

where $\mathbf{p}_t^{(m)} = \mathrm{softmax}(\mathbf{s}^{(t,m)}/\tau_{\text{KD}})$ and $\mathbf{p}_s^{(m)} = \mathrm{softmax}(\mathbf{s}^{(s,m)}/\tau_{\text{KD}})$. Here, $\mathbf{s}^{(s,m)}, \mathbf{s}^{(t,m)} \in \mathbb{R}^N$ denote the student and teacher ranking logits at cardinality m, τ_{KD} is the distillation temperature, and KL denotes Kullback-Leibler divergence.

Representation-level distillation. In addition, we align the semantic tokens between teacher and student. Let $\mathbf{V}_i^{(t)} \in \mathbb{R}^{K \times d}$ and $\mathbf{V}_i^{(s)} \in \mathbb{R}^{K \times d}$ denote the semantic tokens for passage p_i from the teacher and student, respectively. We minimize a Frobenius-norm loss:

$$\mathcal{L}_{\text{KD}}^{\text{repr}} = \frac{1}{NK} \sum_{i=1}^{N} \left\| \mathbf{V}_i^{(t)} - \mathbf{V}_i^{(s)} \right\|_F^2. \quad (8)$$

This encourages the student to learn better representations, helping improve its ranking performance.

Final objective. The full distillation loss is:

$$\mathcal{L}_{\text{KD}} = \mathcal{L}_{\text{KD}}^{\text{logit}} + \beta_{\text{repr}} \mathcal{L}_{\text{KD}}^{\text{repr}}, \quad (9)$$

and the final training objective is:

$$\mathcal{L}_{\text{FLASH-LITE}} = \mathcal{L}_{\text{rank}} + \lambda_{\text{orth}} \mathcal{L}_{\text{orth}} + \lambda_{\text{KD}} \mathcal{L}_{\text{KD}}, \quad (10)$$

where β_{repr}, λ_{orth}, and λ_{KD} denote the weights for representation-level distillation, orthogonal regularization, and overall distillation loss, respectively.

4 Experiments

4.1 Experimental Setup

Datasets. For fair comparison, we use the training, validation, and test data released by MVP (Na et al., 2025), which contains 1M instances, and is derived from MS MARCO dataset (Nguyen et al., 2016). For evaluation, we use the same benchmark setup and candidate sets released by MVP (Na et al., 2025), considering both in-domain and out-of-domain reranking benchmarks. For in-domain evaluation, it uses the TREC Deep Learning 2019 and 2020 passage ranking benchmarks (Craswell et al., 2020). To assess out-of-domain generalization, evaluation is done on the same eight BEIR datasets as MVP, listed in Table 2. BM25 is used as the first-stage retriever, and reranking performance is measured using nDCG@10. We use the publicly available MVP codebase and its released training/validation and evaluation datasets for research purposes only.

Implementation Details. We use T5-base (Raffel et al., 2020), a 220M-parameter encoder–decoder language model, as the backbone for both FLASH and FLASH-LITE. Unless otherwise

Method	TREC DL		BEIR								BEIR	Overall
	DL 19	DL 20	Covid	NFC	Sig.	News	Rob.	Sci.	Tou.	DBP	Avg.	Avg.
MonoT5	71.50	67.00	78.30	35.70	32.00	48.00	53.40	73.10	29.60	42.80	49.10	62.53
RankT5	72.40	68.30	77.70	35.10	30.80	45.40	54.30	73.50	37.10	43.70	49.70	63.47
ListT5	71.80	68.10	78.30	35.60	**33.50**	48.50	52.10	74.10	33.40	43.70	49.90	63.27
PE-Rank	70.80	65.40	77.80	34.80	32.00	**52.30**	48.70	70.20	34.20	40.60	49.00	61.73
MVP	73.45	67.30	**80.06**	**35.90**	31.98	48.31	54.97	**75.92**	33.59	43.51	50.53	63.76
FLASH-L$_1$	72.61$_{\pm 0.62}$	67.79$_{\pm 0.31}$	78.48	35.43	31.62	49.28	52.05	72.70	37.23	42.94	49.97$_{\pm 0.30}$	63.46$_{\pm 0.18}$
FLASH-L$_2$	72.57$_{\pm 0.23}$	67.70$_{\pm 0.25}$	78.70	35.50	31.53	49.33	52.22	72.86	37.55	43.06	50.09$_{\pm 0.30}$	63.45$_{\pm 0.11}$
FLASH-L$_3$	72.54$_{\pm 0.27}$	67.75$_{\pm 0.25}$	78.96	35.55	31.29	49.42	52.21	72.86	37.61	43.08	50.12$_{\pm 0.36}$	63.47$_{\pm 0.07}$
FLASH$_1$	72.44$_{\pm 0.84}$	67.37$_{\pm 1.47}$	79.00	34.91	32.26	48.13	54.76	70.86	37.23	42.46	49.97$_{\pm 0.64}$	63.26$_{\pm 0.93}$
FLASH$_2$	73.56$_{\pm 0.26}$	68.75$_{\pm 0.46}$	79.52	35.03	32.63	48.60	**55.11**	73.69	39.23	43.63	50.93$_{\pm 0.53}$	64.42$_{\pm 0.33}$
FLASH$_3$	**73.89**$_{\pm 0.51}$	**69.25**$_{\pm 0.43}$	79.62	35.14	32.76	48.65	55.05	74.11	**40.21**	**43.98**	**51.19**$_{\pm 0.61}$	**64.78**$_{\pm 0.19}$

Table 2: nDCG@10 (%) across datasets. Results for FLASH and FLASH-LITE (denoted as FLASH-L) are averaged over 5 random seeds. Standard deviations are shown in gray for TREC DL, BEIR average, and overall average. Best mean results are in **bold**. Rows shaded in gray correspond to our methods. Subscripts denote the number of selected semantic tokens m.

Method	Avg. nDCG@10	Latency (s)	Online FLOPs
MVP	63.76	0.360	7.889×10^{14}
FLASH-LITE$_1$	63.46	**0.032**	$\mathbf{5.797 \times 10^{9}}$
FLASH-LITE$_2$	63.45	0.032	8.632×10^{9}
FLASH-LITE$_3$	63.47	0.032	11.467×10^{9}
FLASH$_1$	63.26	0.342	7.454×10^{14}
FLASH$_2$	64.42	0.344	7.457×10^{14}
FLASH$_3$	**64.78**	0.344	7.460×10^{14}

Table 3: Ranking performance and online efficiency comparison for the second stage reranking. Subscripts denote m.

stated, we compare against reranking baselines built on the same T5-base backbone, so that differences in performance and efficiency primarily reflect the reranking architecture rather than model scale. For FLASH-LITE, we use $L_f = 4$ lightweight self-attention layers. Both FLASH and FLASH-LITE use $K = 3$ semantic tokens. For FLASH, we use a learning rate of 2×10^{-5}, batch size 32, gradient accumulation 2, and train for 20 epochs with a linear learning-rate schedule and 5% warmup. We set the softmax temperature to $\tau = 0.8$ and the orthogonality weight to $\lambda_{\text{orth}} = 1.0$. For distillation in FLASH-LITE, we use $\tau_{\text{KD}} = 2$, $\beta_{\text{repr}} = 1.0$, and $\lambda_{\text{KD}} = 1.4$. All hyperparameters are selected based on validation performance. We use TREC-DL21 (Craswell et al., 2021) as the validation set and select the best checkpoint according to nDCG@10.

Experiments are conducted on NVIDIA H100 80GB GPUs. Training uses 8 GPUs, while evaluation is performed on a single GPU. Training FLASH takes approximately 10 hours, whereas training the FLASH-LITE student model with distillation takes approximately 19.5 hours.

Following MVP, we measure efficiency for the second-stage reranking model using the same BM25-retrieved Top-100 candidate sets. FLOPs are measured using DeepSpeed's FLOPs Profiler (Rasley et al., 2020). For FLASH-LITE, both latency and FLOPs include online query encoding, fusion, semantic-token compression, and listwise scoring, while offline passage encoding is excluded.

4.2 Main Results

We run FLASH and FLASH-LITE with five different random seeds and report the mean performance along with standard deviation. We additionally reran MVP across multiple random seeds under the same experimental protocol to assess baseline variability; the reruns yielded similar aggregate performance and did not alter the relative conclusions reported below. Table 2 reports ranking effectiveness on TREC DL19 and DL20 (in-domain), and eight BEIR datasets, which evaluate out-of-domain generalization. Table 3 summarizes the effectiveness–efficiency trade-off in terms of overall average nDCG@10, latency, and FLOPs. Figure 1 further visualizes this trade-off by plotting ranking effectiveness against computational cost.

While the most compact setting with $m = 1$ is less effective than MVP, FLASH outperforms

MVP when using two or three semantic tokens per passage, despite using fewer summary vectors than MVP's four-view representation. In particular, with $m = 3$, it obtains the best overall average nDCG@10 of 64.78%. On BEIR, FLASH$_3$ achieves an average nDCG@10 of 51.19 compared with 50.53 for MVP, showing competitive out-of-domain performance with heterogeneous dataset-level effects. Excluding Touché, the average over the remaining seven BEIR datasets is 52.76 for FLASH$_3$ and 52.95 for MVP, indicating that Touché contributes most of the aggregate BEIR gain.

To evaluate robustness beyond passage-level reranking, we additionally test FLASH$_3$ on the TREC DL20-Document benchmark. FLASH$_3$ achieves 63.912 nDCG@10, compared with 62.647 for MVP, an improvement of 1.265 points. This result indicates that the effectiveness gain is preserved in this substantially longer-input document-reranking setting. Our existing evaluation also includes TREC-NEWS and Robust04, which contain full-length news articles; FLASH$_3$ matches or exceeds MVP on both datasets.

The results across different values of m further demonstrate the benefit of cardinality-configurable inference. For FLASH, increasing the number of selected semantic tokens generally improves ranking performance, with the overall average increasing from 63.26 at $m = 1$ to 64.78 at $m = 3$. This trend suggests that multiple semantic tokens provide additional capacity for preserving ranking-relevant information, while still keeping listwise scoring compact. At the same time, the ability to vary m within a single trained model provides a controllable effectiveness–efficiency trade-off at inference time.

Across five random seeds, FLASH$_3$ is stable overall, achieving 64.78 ± 0.19 average nDCG@10. Its standard deviation is below 1 on DL19, DL20, and six of the eight BEIR datasets, with higher variability concentrated on Signal-1M and Touché. Variability is also greater for the more compressed $m = 1$ setting, indicating that smaller semantic-token budgets are more sensitive to initialization. Per-dataset standard deviations are reported in Appendix C.

The efficiency results in Table 3 illustrate two operating points. FLASH$_3$ achieves higher effectiveness than MVP (64.78 vs. 63.76 average nDCG@10) at approximately the same latency (0.344 vs. 0.360 s/query). FLASH-LITE instead targets low-cost online inference, reducing latency to 0.032 s/query, or 11.25× lower than MVP, at a modest effectiveness trade-off. For FLASH-LITE, the reported online latency and FLOPs exclude offline passage encoding, which is performed once and replaced at query time by reuse of cached passage representations. It achieves an overall average nDCG@10 of 63.47%, matching RankT5 and outperforming MonoT5, ListT5, and PE-Rank, while it is slightly below MVP. Since passage encodings are precomputed offline, FLASH-LITE reduces query-time latency by 11.25× compared with MVP and lowers online FLOPs by approximately 10^5×, substantially lowering the online inference cost. PE-Rank is a close comparison to FLASH-LITE, since it supports offline passage encoding. Despite using a much smaller 220M T5-base backbone than PE-Rank's 7B LLM, FLASH-LITE achieves stronger reranking performance while preserving offline passage encoding. For FLASH-LITE, nDCG@10 changes little across different values of m. This suggests that the primary bottleneck lies before semantic token compression, in the lightweight query–passage fusion stage. Increasing m can preserve more of the fused representation, but cannot recover deep query-conditioned interactions that are not formed by the lightweight fusion layers. Thus, smaller m is a practical choice for efficiency-focused deployments.

Latency does not scale proportionally with FLOPs because changing m mainly affects decoder-side listwise scoring, while runtime is dominated by encoder computation in FLASH, and data movement and fixed GPU/kernel overheads for FLASH-LITE. Thus, FLOP reductions across m are more visible than latency differences in this setting.

4.3 Ablation Studies

4.3.1 Architectural Ablations

We perform architectural ablations to isolate the contributions of STC and the joint decoding mechanism. Replacing MVP's encoder-based multi-view token generation module with STC maintains comparable ranking effectiveness while reducing representation-construction cost. This shows that lightweight attention-based aggregation can serve as an efficient alternative to encoder-side multi-view generation. Next, we replace MVP's independent representation-level scoring with joint decoding. This improves ranking performance by scoring all semantic tokens from all passages in a single de-

coder step, rather than independently scoring each representation and aggregating the results. Detailed results are provided in Appendix B.

4.3.2 Orthogonality Loss and Number of Semantic Tokens

We analyze the effect of orthogonality regularization and the number of semantic tokens used at inference. As shown in Figure 4, with orthogonality, performance peaks at $m = 3$, indicating that three semantic tokens provide a good effectiveness–efficiency trade-off in our benchmark setting. Using more tokens beyond the optimal cardinality does not further improve performance. Since token k participates in the ranking objective only for cardinalities $m \geq k$, earlier tokens receive supervision through more ranking-loss paths and are encouraged to carry stronger ranking signals. Later tokens may therefore remain geometrically distinct while being less ranking-salient.

At inference time, the selected query and passage semantic tokens are uniformly averaged. Although the decoder can downweight a semantic token within the listwise memory, it cannot remove that token's contribution to the averaged query input $\bar{\mathbf{v}}_q^{(m)}$ or to the final passage representation $\bar{\mathbf{v}}_i^{(m)}$. Consequently, adding less ranking-salient tokens can dilute the leading representation as m increases. Larger semantic-token budgets may nevertheless be useful for more complex retrieval settings, such as longer documents or larger candidate sets.

Without orthogonality regularization, the model requires more tokens to improve performance (peak performance at $m = 8$). Even with more tokens, it does not match the performance of the orthogonally regularized model. Direct measurements of the semantic-token geometry further confirm the intended effect of the orthogonality objective. At the main operating point ($K = 3$), the mean absolute off-diagonal cosine similarity decreases from 0.9807 without $\mathcal{L}_{\text{orth}}$ to 0.0060 with $\mathcal{L}_{\text{orth}}$, while the effective rank increases from 1.0702 to 2.9998. This confirms that the selected three-token representation does not collapse to highly redundant vectors. Detailed geometric measurements are provided in Appendix B.

4.3.3 Cardinality-Configurable Learning

We evaluate the effect of training across multiple semantic-token cardinalities. Removing this objective reduces overall nDCG@10 from 64.74 to 63.77, showing that multi-cardinality training improves robustness under different inference-time token budgets.

We further examine the interaction between candidate-set size N and semantic-token budget m. For small candidate sets ($N = 20$), the best and worst token budgets differ by only 0.182 nDCG@10. In contrast, the gain from increasing m from 1 to 3 grows from 0.102 at $N = 20$ to 3.209 at $N = 100$. Thus, smaller candidate sets can operate with a cheaper token budget with little loss, whereas larger candidate sets benefit more from additional semantic tokens. This further motivates multi-cardinality training, which allows the same checkpoint to adapt its token budget at inference time without retraining. Full results are provided in Appendix B.

4.3.4 Knowledge Distillation in FLASH-LITE

We further evaluate the complementary contributions of logit-level and representation-level distillation in FLASH-LITE. Without distillation, overall nDCG@10 is 57.71. Adding logit-level distillation improves overall nDCG@10 to 62.54 by transferring the teacher's ranking distribution at each cardinality. Adding representation-level distillation further improves performance to 63.44. A similar trend is observed on BEIR, where average nDCG@10 increases from 43.03 without distillation to 48.34 with logit-level distillation and to 50.06 with both objectives. Thus, logit-level distillation recovers most of the performance loss, while representation-level alignment provides a further consistent gain. Detailed results are provided in Appendix B.

5 Conclusion

We introduced FLASH, an efficient listwise reranker that represents query–passage pairs with compact semantic tokens and jointly scores candidates in a single decoder step. We also proposed cardinality-configurable training for flexible inference and FLASH-LITE, which enables offline passage encoding for low-cost online reranking. Experiments on TREC and BEIR show that FLASH improves ranking effectiveness at comparable online inference cost, while FLASH-LITE achieves substantial online efficiency gains with competitive performance.

6 Limitations

This work focuses on improving the efficiency of listwise reranking by replacing token-heavy pas-

sage representations with compact semantic tokens. While this design substantially reduces the amount of tokens passed to the decoder, it also introduces a compression bottleneck.

Following existing reranking work, we use BM25 as the first-stage retriever and focus on second-stage reranking. All methods are evaluated on the same BM25-retrieved candidate sets, ensuring that performance differences are attributable to the reranker rather than the retriever.

Our evaluation focuses on standard passage reranking benchmarks using English text datasets. Extending FLASH to multilingual and cross-language retrieval, such as NeuCLIRBench (Lawrie et al., 2025), is an important direction for future work.

Developing adaptive strategies that select m dynamically based on query difficulty, passage complexity is an important direction for future work. Our current scoring formulation also uniformly aggregates selected semantic tokens and does not include learned token gating or end-to-end masking; adaptive weighting of semantic tokens is an important direction for future work.

References

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2021. Overview of the TREC 2020 deep learning track. In *Proceedings of TREC*.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. 2020. Overview of the TREC 2019 deep learning track. In *Proceedings of TREC*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of NAACL-HLT*, pages 4171–4186.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of EMNLP*, pages 6769–6781.

Dawn Lawrie, James Mayfield, Eugene Yang, Andrew Yates, Sean MacAvaney, Ronak Pradeep, Scott Miller, Paul McNamee, and Luca Soldani. 2025. Neuclir-bench: A modern evaluation collection for monolingual, cross-language, and multilingual information retrieval. *arXiv preprint arXiv:2511.14758*.

Qijiong Liu, Jieming Liu, Conghui Wu, Lu Fan, and Xiao-Ming Wu. 2024. PE-Rank: Leveraging LLM passage embeddings for listwise reranking. *arXiv preprint arXiv:2406.14458*.

Jeongwoo Na, Jun Kwon, Eunseong Choi, and Jongwuk Lee. 2025. Multi-view-guided passage reranking with large language models. *arXiv preprint arXiv:2509.07485*.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. In *Proceedings of the NeurIPS Workshop on Cognitive Computation*.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with BERT. *arXiv preprint arXiv:1901.04085*.

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document ranking with a pre-trained sequence-to-sequence model. In *Findings of EMNLP*, pages 708–718.

Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. RankVicuna: Zero-shot listwise document reranking with open-source large language models. *arXiv preprint arXiv:2309.15088*.

Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2024. RankZephyr: Effective and robust zero-shot listwise reranking is a breeze! *arXiv preprint arXiv:2312.02724*.

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

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 3505–3506.

Ganesh Jawahar Reddy, Sumit Bhatia, Preethi Jyothi, and Sunita Sarawagi. 2024. FIRST: Faster improved listwise reranking with single token decoding. In *Proceedings of EMNLP*.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. In *Foundations and Trends in Information Retrieval*, volume 3, pages 333–389.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *Proceedings of EMNLP*, pages 14918–14937.

Manveer Singh Tamber, Ronak Pradeep, and Jimmy Lin. 2023. Scaling down, LiTting up: Efficient zero-shot listwise reranking with T5. *arXiv preprint arXiv:2312.01514*.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *Proceedings of ICLR*.

Soyoung Yoon, Eunbi Hwang, Jeongyeon Cho, Suyeon Ham, and Moontae Lee. 2024. Listwise reranking with fusion-in-decoder. In *Proceedings of ACL*.

Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. 2023. RankT5: Fine-tuning T5 for text ranking with ranking losses. In *Proceedings of SIGIR*, pages 2308–2313.

A FLASH-LITE Illustration

Figure 3 provides an expanded illustration of the FLASH-LITE architecture. Unlike FLASH, FLASH-LITE separates query and passage encoding to enable offline passage precomputation, followed by lightweight online fusion before semantic token compression and listwise scoring.

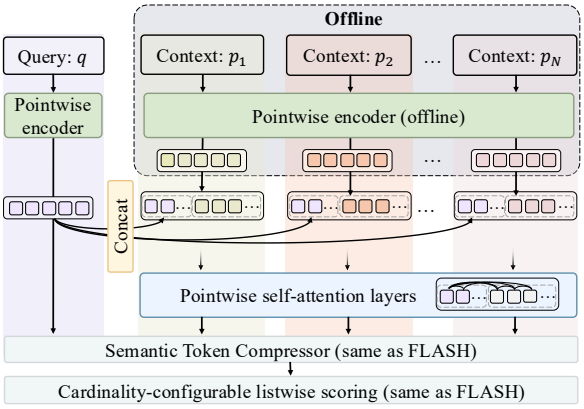


Figure 3: Overview of FLASH-LITE. In contrast to FLASH, FLASH-LITE decouples query and passage encoding, allowing passage representations to be precomputed offline and fused with the query online through lightweight self-attention before semantic token compression and listwise scoring.

B Detailed Ablation Results

This section provides detailed ablation results.

Table 4 provides detailed architectural ablations for the Semantic Token Compressor (STC) and joint decoding mechanism.

Method	Avg. nDCG@10	Avg. FLOPs
MVP	63.76	7.89×10^{14}
MVP + STC	63.91	$\mathbf{7.39 \times 10^{14}}$
MVP + Joint Decoding	64.40	7.89×10^{14}
FLASH_3	**64.78**	7.46×10^{14}

Table 4: Architectural ablations showing the effect of replacing MVP's encoder-side view construction with STC and replacing independent view scoring with joint listwise decoding. Results are reported using average nDCG@10 and average FLOPs.

Figure 4 illustrates the effect of orthogonality regularization and the number of semantic tokens used at inference.

Setting	DL19	DL20	BEIR	Overall
None	67.26	62.86	43.03	57.71
Logit	71.96	67.31	48.34	62.54
Logit + Representation	72.26	68.01	50.06	63.44

Table 6: Component-wise distillation ablation for FLASH-LITE. Logit distillation recovers most of the performance loss, while representation-level alignment provides a further consistent gain. Results are reported using nDCG@10.

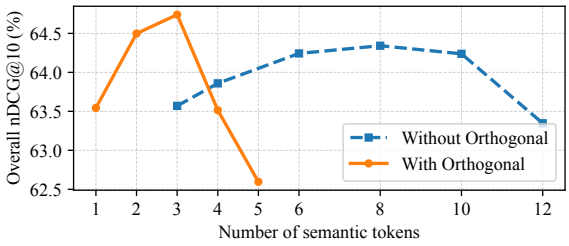


Figure 4: Effect of orthogonality regularization and number of semantic tokens m on overall nDCG@10.

Table 5 reports the effect of cardinality-configurable training, and Table 6 reports the effect of knowledge distillation for FLASH-LITE.

Method	DL19	DL20	BEIR	Overall
w/o Card. Training	72.34	68.33	50.64	63.77
w/ Card. Training	**73.67**	**69.39**	**51.16**	**64.74**

Table 5: Effect of the proposed cardinality-configurable learning objective. Results are reported using nDCG@10.

Candidate-Set Size and Semantic-Token Budget. We study the interaction between candidate-set size N and the number of selected semantic tokens m on DL19 and DL20. We evaluate $N \in \{20, 50, 75, 100\}$ and $m \in \{1, 2, 3\}$. Table 7 reports nDCG@10 for each configuration.

At $N = 20$, the best and worst configurations differ by only 0.182 nDCG@10. Relative to $m = 1$, increasing the token budget to $m = 3$ improves the average DL nDCG@10 by 0.102, 0.441, 2.695, and 3.209 for $N = 20$, 50, 75, and 100, respectively. These results indicate that smaller candidate pools can use a lower semantic-token budget with negligible loss, whereas larger pools benefit substantially more from additional semantic tokens.

Prefix Selection and Token Ordering. Although semantic-token order has no predefined meaning,

N	DL19			DL20			Average		
	$m=1$	$m=2$	$m=3$	$m=1$	$m=2$	$m=3$	$m=1$	$m=2$	$m=3$
20	64.765	64.876	64.891	60.933	61.185	61.011	62.849	63.031	62.951
50	70.399	70.233	70.984	66.360	66.618	66.656	68.380	68.426	68.820
75	71.282	72.952	73.613	65.589	67.931	68.648	68.436	70.442	71.131
100	71.641	73.686	74.514	65.496	68.224	69.041	68.569	70.955	71.778

Table 7: Interaction between candidate-set size N and semantic-token budget m. Values are nDCG@10. The benefit of larger semantic-token budgets increases with candidate-set size.

Selected Tokens	Avg. DL nDCG@10
$\{1\}$	**71.819**
$\{2\}$	70.360
$\{3\}$	68.569
$\{1,2\}$	**71.822**
$\{1,3\}$	71.570
$\{2,3\}$	70.955

Table 8: Effect of semantic-token subset selection at Top-100. The learned prefix achieves the highest effectiveness for both one- and two-token selection.

Metric	w/o $\mathcal{L}_{\text{orth}}$	w/ $\mathcal{L}_{\text{orth}}$
Mean absolute off-diagonal cosine ↓	0.9807	0.0060
Effective rank ↑	1.0702	2.9998
$\|\hat{\mathbf{V}}\hat{\mathbf{V}}^\top - \mathbf{I}\|_F^2$ ↓	5.7743	0.0004

Table 9: Geometry of passage semantic tokens immediately after STC, measured over DL19 and DL20 queries. Orthogonality regularization substantially reduces inter-token similarity and increases effective rank.

the multi-cardinality objective induces an asymmetric training signal: token 1 participates in the ranking objective for every cardinality, token 2 for $m \geq 2$, and token 3 only for $m = 3$. We evaluate whether this learned ordering makes prefix selection preferable by exhaustively testing all subsets of the three semantic tokens at Top-100. Table 8 reports the average nDCG@10 over DL19 and DL20.

The learned prefix is best for both $m = 1$ and $m = 2$. Since all possible subsets at these cardinalities are evaluated, this analysis also exhaustively covers random subset choices and indicates that multi-cardinality training induces a useful ordering in which the leading semantic tokens are generally more informative. A query-dependent token selector could potentially provide additional flexibility, but would require an additional selection module and training objective, whereas prefix selection remains parameter-free.

Semantic-Token Geometry. To directly evaluate whether the orthogonality objective prevents semantic-token collapse, we measure the geometry of passage semantic tokens immediately after the Semantic Token Compressor (STC) over DL19 and DL20 queries. Table 9 compares models trained with and without $\mathcal{L}_{\text{orth}}$ at $K = 3$.

The paired 95% confidence intervals for the changes from without to with $\mathcal{L}_{\text{orth}}$ are $[-0.9767, -0.9728]$ for mean absolute off-diagonal cosine, $[1.9236, 1.9355]$ for effective rank, and $[-5.7959, -5.7516]$ for the orthogonality error, respectively. These measurements directly verify that $\mathcal{L}_{\text{orth}}$ prevents collapse of the selected three-token representation.

C Per-Dataset Variability

Table 10 reports the standard deviation of nDCG@10 across five random seeds for each evaluation dataset. FLASH_3 exhibits low variability overall (± 0.19), with standard deviation below 1 on DL19, DL20, and six of the eight BEIR datasets. Higher variability is concentrated on Signal-1M and Touché. The more compressed $m = 1$ configuration is generally more sensitive across seeds, with an overall standard deviation of 0.93 compared with 0.33 and 0.19 for $m = 2$ and $m = 3$, respectively.

C.1 Candidate-Replacement Analysis

To directly test whether FLASH scores depend on the composition of the candidate set, we conduct a controlled candidate-replacement analysis on DL20. For each query, we consider the Top-10 candidates and replace one candidate at a time, in the same rank position, with 10 randomly sampled candidates from ranks 11–100. The remaining nine candidates and their semantic representations are left unchanged. Across 54 queries, 10 replacement positions, and 10 sampled replacements per position, this results in 5,400 interventions.

Replacing a candidate changes the scores of the nine retained candidates by an average abso-

Method	TREC DL		BEIR								BEIR Avg.	Overall
	DL19	DL20	Covid	NFC	Sig.	News	Rob.	Sci.	Tou.	DBP		
FLASH_1	0.84	1.47	0.69	0.57	0.87	0.69	0.77	2.77	1.83	1.40	0.64	0.93
FLASH_2	0.26	0.46	0.85	0.43	1.18	0.59	0.79	1.15	1.76	0.67	0.53	0.33
FLASH_3	0.51	0.43	0.72	0.44	1.17	0.66	0.98	0.97	2.13	0.43	0.61	0.19

Table 10: Standard deviation of nDCG@10 across five random seeds. FLASH_3 shows low variability on most datasets, with higher variance concentrated on Signal-1M (Sig.) and Touché (Tou.).

lute amount of 0.212, with a query-bootstrap 95% confidence interval of $[0.188, 0.237]$. To separate candidate-specific effects from a common shift in all logits, we additionally center the score changes within each intervention; the mean absolute centered change remains 0.129. Pairwise score margins between retained candidates change by 0.183 on average.

The effect also depends on the position of the replaced candidate. Replacing the rank-1 candidate produces an average pairwise-margin change of 0.247, compared with 0.213 for replacements at ranks 2–5 and 0.147 for ranks 6–10. Because the semantic representations of the retained candidates are fixed during these interventions, these off-target score changes propagate through the shared listwise decoder state. This provides direct evidence that FLASH scoring is candidate-set dependent rather than independently pointwise.

AI assistants were used for writing and limited coding assistance.