Learning Right Monotone Permutation Matrices for Neural Subsequence Search

Bhavya Kohli[†] **Soutrik Sarangi**[*] **Aziz Shameem**[*] **Abir De**
Indian Institute of Technology Bombay

Abstract

Subsequence retrieval seeks relevant segments in a large corpus given a short query. Existing pairwise metric-based methods are computationally intensive, hard to parallelize, and tied to domain-specific metrics. In this work, we introduce a neural framework that casts subsequence matching as end-to-end alignment with permutation matrices satisfying monotonicity used as differentiable approximate subsequence selectors. Our framework yields fixed-dimensional embeddings for variable-length inputs, and we prove these embeddings are compatible with standard Approximate Nearest Neighbor search methods such as Locality-sensitive hashing (LSH), enabling scalable retrieval. We also impose structural priors on admissible subsequences and integrate them directly into the scoring function. The approach is domain-agnostic and operates on pre-trained representations across modalities. Experiments on real-world datasets from two different domains show strong retrieval performance and substantial speedups, with high parallelism on GPU-accelerated hardware.

1 INTRODUCTION

Subsequence retrieval—the task of finding sequences in a corpus that contain a given query—is fundamental to applications ranging from bioinformatics (Ibrahim et al., 2022; Tahir et al., 2017; Montanari et al., 2016; Daugelaite et al., 2013; Smith and Waterman, 1981), text-snippet search (Pasternack and Roth, 2009; Baeza-Yates, 1991), image sequence retrieval (Park and Chu, 2003; Yazdani and Ozsoyoglu, 1996), to time-series data mining (Rakthanmanon et al., 2012; Shieh and Keogh, 2008), *etc.*

Classical approaches often compute distances based on Dynamic Time Warping (DTW) (Van Do and Anh, 2017; Rakthanmanon et al., 2012; Zhou and Wong, 2008; Berndt and Clifford, 1994) or its variants (Cao et al., 2025; Han and Zhang, 2022; Zhao and Itti, 2018; Boulnemour and Boucheham, 2018; Papapetrou et al., 2011). However, DTW relies on dynamic programming (DP) to find an optimal alignment, a process that is inherently sequential and must be re-solved for every query-corpus pair, thus limiting scalability. While significant advances have introduced approximations for speed as in (Salvador and Chan, 2007), which employs a multilevel approximation strategy, or (Blondel et al., 2021; Cuturi and Blondel, 2017), which add differentiability for gradient-based learning, they do not overcome the core bottlenecks. Specifically, DTW-based methods suffer from three key limitations: **(I)** Their recursive computation precludes parallelization *along* the sequence axis, hindering efficient GPU usage. **(II)** The resulting alignment-based distances are incompatible with standard Approximate Nearest Neighbor (ANN) indexing. **(III)** They are designed for contiguous subsequence alignment rather than the more general task of arbitrary subsequence matching.

1.1 Present Work

We introduce a neural framework for subsequence matching–based sequence retrieval that addresses the above limitations. Our contributions are:

Subsequence Search as Optimization over Right Monotone Permutations We pose ordered subsequence search as a constrained linear assignment problem (CLAP) that minimizes a transport-based *relevance distance*. To enforce order, we restrict permutations to right monotone matrices, whose column in-

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

[*]Equal contribution.
[†]Correspondence to bhavyakohli@u.nus.edu.

dices (corpus positions) increase monotonically across rows (query positions). This yields an order-preserving alignment naturally suited to differentiable modeling.

Trainable Score to Guide Constraint Satisfaction Viewing each row of a permutation-matrix as a one-hot encoding of its active column index lets us express right-monotonicity as a set of linear constraints. We integrate these constraints into the score via Lagrange-multiplier-based penalties proportional to violations of the same.

Neural Subsequence Matching Model Our model approximates the CLAP variables: the permutation matrix and the Lagrangian-multiplier vector. The permutation is relaxed to a doubly stochastic (soft) matrix via a three-stage differentiable alignment process: **(i)** a transformer encoder that produces contextual embeddings for the query and candidate corpus sequences; **(ii)** the construction of a specially designed reward matrix, followed by **(iii)** iterative Sinkhorn normalization (Mena et al., 2018; Sinkhorn, 1964) of the reward matrix which yields a soft permutation. A companion network with stacked convolutions, adaptive pooling, and a positive-activation head produces the nonnegative multipliers. The design enables high parallelism and efficient GPU execution, hence avoiding the efficiency pitfall of the metric-based baselines.

Efficient Retrieval with Guarantees We define relevance using an Earth mover's distance (EMD) metric induced by a right monotone permutation matrix, and extend the work in Davidson and Dym (2024) by training sequence-level encoders whose outputs are Lipschitz with respect to EMD, preserving EMD margins under the Euclidean norm. For fast lookup, we learn a hyperplane-based LSH scheme (Liu et al., 2014, 2011). Unlike Roy et al. (2023), which enforces only *first-order* balance, we add a *second-order* loss that balances outer products of hash bits, promoting uniformity across $(+1,+1)$, $(-1,-1)$, $(+1,-1)$, and $(-1,+1)$ hash bit pairs and yielding more informative, evenly distributed codes.

Across diverse datasets with real-world imperfections, our method achieves higher retrieval performance and substantially lower runtime than existing baselines.

2 PRELIMINARIES

Notation We write $s^{(q)} = (x_1^{(q)}, \ldots, x_m^{(q)})$ as a query sequence and $s^{(c)} = (x_1^{(c)}, \ldots, x_n^{(c)})$, with $n > m$, as a corpus sequence with elements $x_\bullet^{(\bullet)} \in \mathcal{X}$. We collect their contextual embeddings $\boldsymbol{x}_i^{(q)}, \boldsymbol{x}_j^{(c)} \in \mathbb{R}^{\dim_x}$ into matrices $\boldsymbol{X}^{(q)}$ and $\boldsymbol{X}^{(c)}$, where $\boldsymbol{X}^{(q)} \in \mathbb{R}^{m\times\dim_x}$ and $\boldsymbol{X}^{(c)} \in \mathbb{R}^{n\times\dim_x}$. We write the set of corpus sequences as $C = \{s^{(c)}\}$. We denote $\mathcal{P}_n$ and $\mathcal{B}_n$ as the sets of all $n \times n$ permutation and doubly stochastic (soft permutation) matrices, and write $\boldsymbol{P}$ for both hard and soft permutation matrices (See Appendix D for formal definitions of $\mathcal{P}_n$ and $\mathcal{B}_n$). Let $\{\boldsymbol{e}_0^{(k)}, \ldots, \boldsymbol{e}_{k-1}^{(k)}\}$ be the canonical basis of $\mathbb{R}^k$. We define $\boldsymbol{S} \in \mathbb{R}^{m\times n}$ as a m-row-selector matrix with $\boldsymbol{S}[i,:] = \boldsymbol{e}_i^{(n)}$ so that for $\boldsymbol{L} \in \mathbb{R}^{n\times n}$, $\boldsymbol{SL} = \boldsymbol{L}[:m,:]$, and define $\boldsymbol{A} \in \mathbb{R}^{m\times m}$ with $\boldsymbol{A}[0,:] = \boldsymbol{0}_m$ and $\boldsymbol{A}[i,:] = \boldsymbol{e}_i^{(m)} - \boldsymbol{e}_{i-1}^{(m)}$ for $1 \le i < m$.

Subsequence Matching Given $s^{(q)}$ and $s^{(c)}$ as defined above, $s^{(c)}$ is relevant to $s^{(q)}$ if an m-length *ordered subsequence* of $s^{(c)}$ is similar to $s^{(q)}$, i.e., there exists $t = (x_{\pi_1}^{(c)}, \ldots, x_{\pi_m}^{(c)}) \subset s^{(c)}$ with $x_{\pi_i}^{(c)} \approx x_i^{(q)}$ in $\mathcal{X}$ such that the order is preserved, i.e., $\pi_j > \pi_{j+1}$ for all j. We denote the binary relevance of c to q by $y(q,c) \in \{0,1\}$, and write $C_{q\checkmark} = \{c \,|\, y(q,c) = 1\}$ and $C_{q\times} = \{c \,|\, y(q,c) = 0\}$.

Our Goal Given training queries $Q = \{s^{(q)}\}$, a corpus C, and labels $y_{qc} \in \{0,1\}$ for $(q,c) \in Q \times C$, our goals are: **(I)** learn a subsequence-matching retrieval framework which, when trained on $D = (Q, C, \{y_{qc}\})$, returns the top-k relevant corpus sequences for an unseen q' accurately and efficiently; **(II)** design a trainable hyperplane-based hashing method to index C into hash buckets so that for any q', top-k retrieval runs in $o(|C|)$ time.

3 PROPOSED APPROACH

In this section, we represent the subsequence search problem as a CLAP and derive a soft-permutation solution to it. Then, we design our framework OPAS, for Ordered Permutation-Guided Alignment for Subsequence Matching, building upon this formulation.

Alignment Using Permutation Matrices In OPAS, we first employ an embedding function f_{embed} to capture patterns and nuances in the original query and corpus sequences to generate dense contextual embeddings $\boldsymbol{X}^{(q)}$ and $\boldsymbol{X}^{(c)}$. This allows semantically similar sequences to be close in latent space, and ensures that sequences which are seemingly different but still relevant are also retrieved. We work exclusively with these embeddings for our optimization. For a permutation matrix $\boldsymbol{P} \in \mathcal{P}_n$, we consider the quantity $\boldsymbol{PX}^{(c)}$ as a new sequence created by shuffling the rows in $\boldsymbol{X}^{(c)}$, and the quantity $\boldsymbol{SPX}^{(c)}$ as the first m elements of this new sequence. We view this combined process as the selection of an m-length unordered subsequence from $\boldsymbol{X}^{(c)}$ using an alignment matrix $\boldsymbol{P}$. We moti-

vate and discuss our choice of the embedding function f_{embed} in Section 3.1.

Finding the Closest Unordered Subsequence In this case, we simply desire an optimal permutation matrix $\boldsymbol{P}^*$ such that $\|\boldsymbol{X}^{(q)} - \boldsymbol{S}\boldsymbol{P}^*\boldsymbol{X}^{(c)}\|_F^2$ is minimum:

$$\boldsymbol{P}^* = \underset{\boldsymbol{P}\in\mathcal{P}_n}{\arg\min} \|\boldsymbol{X}^{(q)} - \boldsymbol{S}\boldsymbol{P}\boldsymbol{X}^{(c)}\|_F^2 \tag{1}$$

Denoting $\langle \boldsymbol{P}, \boldsymbol{Z}\rangle_F = \text{tr}(\boldsymbol{P}\boldsymbol{Z}^\top)$ as the Frobenius inner product, we may also rewrite (see Equation 16 in Appendix E):

$$\boldsymbol{P}^* = \underset{\boldsymbol{P}\in\mathcal{P}_n}{\arg\max} \langle \boldsymbol{P}, \boldsymbol{S}^\top\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}\rangle_F \tag{2}$$

This makes $\boldsymbol{P}^*$ an instance of a combinatorial linear assignment problem (Kuhn, 1955) with reward matrix $\boldsymbol{Z} = \boldsymbol{S}^\top\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}$. We approximate this solution as a $(\tau \to 0^+)$ limit of the Sinkhorn operator (Mena et al., 2018), and use the *incomplete* Sinkhorn operator (Adams and Zemel, 2011) with a finite number of iterations.

Representing an Ordered Subsequence as a Right Monotone Permutation Matrix Let $(r_1, \cdots, r_\ell)$ be a subsequence of a corpus sequence. Denoting α_i as the corresponding index of r_i in the corpus sequence, the ordering constraint requires that $\alpha_i < \alpha_{i+1}\ \forall i$. In OPAS, we extract subsequences indexed by $\boldsymbol{SP}$. Extracting an ordered subsequence requires the bit positions of the successive rows of $\boldsymbol{P}$ (till row m) to move from left to right. More formally, to satisfy the ordering constraint, $\boldsymbol{P} \in \mathcal{P}_n$ must satisfy: for any $i \in [m]$, $P[i][j] = 1 \implies P[k][\ell] = 0\ \forall k < i, \ell > j$. This is our definition of a *right monotone* permutation matrix.

Extracting Bit Positions from Permutation Matrices For row $P[i]$ of a permutation matrix $\boldsymbol{P}$, the bit position can be extracted via $\sum_{j=0}^{N-1} P[i][j]v^j$, where $v > 0$ is a positive real number. Thus, defining $\boldsymbol{a} \in \mathbb{R}^n$ with $a[j] = v^j$, we encode the bit positions of all rows of $\boldsymbol{P}$ in $\boldsymbol{Pa}$.

Using Successive Differences to Ensure Right Monotonicity We truncate $\boldsymbol{Pa}$ to its first m elements by left multiplication with $\boldsymbol{S}$. Define successive differences $\text{diff}_{\boldsymbol{P}}(i)$ (for $i \geq 1$) as $\text{diff}_{\boldsymbol{P}}(i) = (\boldsymbol{SPa})[i] - (\boldsymbol{SPa})[i-1]$. Since we want $\boldsymbol{P}$ to be right monotone, cases where this difference is less than a desired margin b constitute an *order violation*. We augment Equation 1 with parameters $\lambda_j > 0$ (akin to Lagrange multipliers) as per-row importances for the right monotone constraint, yielding:

$$\boldsymbol{P}^* = \underset{\boldsymbol{P}\in\mathcal{P}_n}{\arg\min} \|\boldsymbol{X}^{(q)} - \boldsymbol{S}\boldsymbol{P}\boldsymbol{X}^{(c)}\|_F^2 + \sum_{j=1}^{m-1} \lambda_j(b - \text{diff}_{\boldsymbol{P}}(j)) \tag{3}$$

For the λ_j's, we consider two approaches: a neural parameterization dependent on $\boldsymbol{X}^{(q)}$ and $\boldsymbol{X}^{(c)}$, and a constant setting; we compare both in Section 4.3.

Main Optimization Objective Using $\boldsymbol{A}$, we compute $\boldsymbol{ASPa} = \begin{bmatrix}0 & \text{diff}_{\boldsymbol{P}}(1) & \dots & \text{diff}_{\boldsymbol{P}}(m-1)\end{bmatrix}^\top \in \mathbb{R}^m$. Define:

$$\boldsymbol{\lambda} = \begin{bmatrix}0, \lambda_1, \dots, \lambda_{m-1}\end{bmatrix}^\top, \boldsymbol{b} = \begin{bmatrix}b, \cdots, b\end{bmatrix}^\top \in \mathbb{R}^m$$

where b is the scalar fixed margin. Then the objective in Equation 3 becomes:

$$\boldsymbol{P}^* = \underset{\boldsymbol{P}\in\mathcal{P}_n}{\arg\min} \|\boldsymbol{X}^{(q)} - \boldsymbol{S}\boldsymbol{P}\boldsymbol{X}^{(c)}\|_F^2 + \langle \boldsymbol{\lambda},\ \boldsymbol{b} - \boldsymbol{ASPa}\rangle \tag{4}$$

We show next that this combinatorial problem over $\mathcal{P}_n$ is the (τ-temperature) limit of a relaxed optimization over $\mathcal{B}_n$ computed via Sinkhorn iterations.

Theorem 1. *The solution $\boldsymbol{P}^*$ which minimizes the RHS of 4 is:*

$$\begin{aligned}\boldsymbol{P}^* &= \lim_{\tau\to 0^+} S\left(\frac{\boldsymbol{Z}}{\tau}\right) \\ &= \lim_{\tau\to 0^+} S\left(\frac{\boldsymbol{S}^\top\left(2\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top} + \boldsymbol{A}^\top\boldsymbol{\lambda}\boldsymbol{a}^\top\right)}{\tau}\right)\end{aligned} \tag{5}$$

Proof. See Appendix E. □

3.1 Design of OPAS

Overview of Components In OPAS, we design two primary trainable components to address the subsequence retrieval task. First, a neural network Embed_ϕ which performs the role of the sequence embedding function f_{embed} described earlier– transforming input query and corpus sequences ($s^{(q)}$ and $s^{(c)}$) into their corresponding contextual embeddings ($\boldsymbol{X}^{(q)}$ and $\boldsymbol{X}^{(c)}$). Second, a scoring function, Score_β (abbreviated as s), computes the final relevance score for a given query-corpus pair through a parameterized combination of two critical components. When the weighting factors λ_j are parameterized, we introduce a convolution-based network, LamModel_θ, which generates a unique $\boldsymbol{\lambda}$ for each query-corpus pair. Additionally, when the input sequences are not in vector form, we employ a separate neural network, PreEmbed_ω, to first embed individual sequence elements into vectors *before* proceeding with the training of OPAS. Details regarding the architecture and selection of PreEmbed_ω for image sequence datasets are provided in Appendix H.

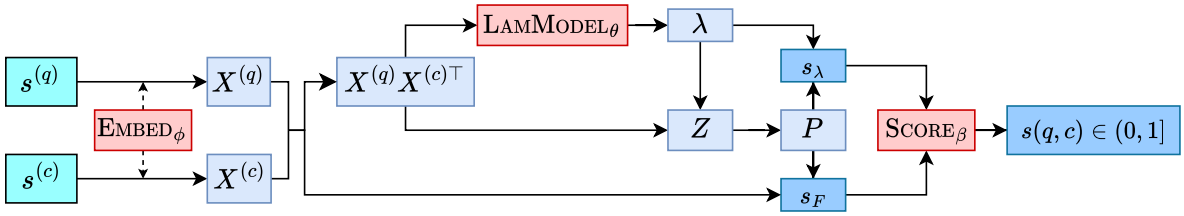

Figure 1: Score computation using OPAS. The dotted lines for EMBED$_\phi$ signify shared weights when embedding a given query and corpus sequence. Note that LAMMODEL$_\theta$ here is replaced with a function returning a fixed constant vector if λ_j's are not being learned.

Choice of Embed$_\phi$ As discussed by Yun et al. (2019), transformers are universal approximators of sequence-to-sequence functions. Given their results and the performance of transformers on sequence based tasks, we chose the transformer encoder (Vaswani et al., 2017) as a trainable f_{embed}. In Section 4.3, we experiment with non-sequential neural models as an alternative choice of f_{embed} to motivate our design choice of using an architecture that is able to capitalize on latent contextual information in input sequences. We use the same network to embed both query and corpus sequences, effectively sharing the parameters of EMBED$_\phi$.

Choice of LamModel$_\theta$ The weighting factors λ_j play a vital role in incorporating the ordering constraint into the reward matrix $\boldsymbol{Z}$ that defines the alignment we obtain for a given query-corpus pair. To ensure that the similarity of query and corpus sequence elements is taken into account when parameterizing these factors, we use the pairwise dot-product similarity matrix $\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}$ as the input to LAMMODEL$_\theta$. We use a series of convolutional layers with kernels acting along rows, which utilize the similarity information encoded in $\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}$. Following the convolutions, we use adaptive pooling and a feedforward network to obtain m parameters, i.e., one weighting factor for each row of the input matrix, essentially aimed at penalizing the successive difference $\text{diff}_{\boldsymbol{P}}(i)$ for that row. In order to ensure these factors are nonnegative, the output is passed through the sigmoid activation function. Out of the m values, only the final $m-1$ are actually relevant since $\text{diff}_{\boldsymbol{P}}(i)$ is defined only for $i \geq 1$, and the first number is zeroed out to finally obtain $\boldsymbol{\lambda}$.

Soft Alignments using Sinkhorn Iterations After constructing the reward matrix $\boldsymbol{Z}$ defined in Theorem 1, we perform successive temperature-dependent sinkhorn normalizations (Mena et al., 2018; Cuturi, 2013) by iteratively normalizing the rows and columns of $\exp(\boldsymbol{Z}/\tau)$ a fixed number of times. We thus obtain a soft permutation matrix $\boldsymbol{P}$ which serves as an approximation of the ideal $\boldsymbol{P}^*$ in 4.

Constituents of the Final Score and Score$_\beta$ To encapsulate both the quality of subsequence alignment and the extent to which the learned alignment satisfies the ordering constraint, we introduce two distinct scoring components. For notational consistency, we view $\boldsymbol{P}$ as a function of q and c in the expression for s_λ.

$$s_F(q,c) = -\|\boldsymbol{X}^{(q)} - \boldsymbol{P}\boldsymbol{X}^{(c)}\|_F^2 \tag{6}$$

$$s_\lambda(q,c) = -\langle \boldsymbol{\lambda}, \boldsymbol{b} - \boldsymbol{A}\boldsymbol{S}\boldsymbol{P}\boldsymbol{a} \rangle \tag{7}$$

We define two strictly positive trainable parameters β_1 and β_2 (See Appendix L.6), and finally define the score aggregation function SCORE$_\beta$ in Equation 8, where $\sigma(\cdot)$ refers to the sigmoid activation function.

$$\begin{aligned} s(q,c) &= \text{SCORE}_\beta(q,c) \\ &= \sigma\left(\beta_1 \cdot s_F(q,c) + \beta_2 \cdot s_\lambda(q,c)\right) \end{aligned} \tag{8}$$

3.2 Training Regime

For a batch of queries $\mathcal{Q} = \{s^{(q)}\}$ and a corresponding batch of corpus sequences $C' \subset C, |\mathcal{Q}| = |C'| = B$, we compute their relevance scores following the functional flow depicted in Figure 1. Direct supervision using a gold label alignment is not practical for evaluating scores, and since our end goal is to rank corpus items at inference time, the use of a hinge-based margin loss is the ideal choice. For this, we use the pairwise difference of scores assigned to positive and negative query-corpus pairs identified using the relevance labels, i.e., whether $y(q,c)$ is 1 or 0 for a given pair. With the margin Δ as a tunable hyperparameter, the loss is defined for a batch as $\mathcal{L}(\mathcal{Q}, C') = \sum_{s^{(q)} \in \mathcal{Q}} \mathcal{L}(q, C')$, where,

$$\mathcal{L}(q,C') = \sum_{\substack{s^{(c_+)} \in C'_{q\checkmark} \\ s^{(c_-)} \in C'_{q\times}}} \left(\Delta + s(q,c_-) - s(q,c_+)\right) \tag{9}$$

3.3 OPAS for LSH

For using embeddings learned by OPAS for LSH, we train an aggregation model to convert sequential embeddings from EMBED$_\phi$ into dense vectors which can be used for indexing and in performing ANN search.

Neural Set-Aggregation on Trained Embeddings We introduce a neural network SetAggr_γ which aggregates a given query (corpus) sequence into a single dense vector $\boldsymbol{h}^{(q)}$ ($\boldsymbol{h}^{(c)}$) $\in \mathbb{R}^{d_{\text{aggr}}}$. After training Embed_ϕ, we obtain embeddings $\boldsymbol{X}^{(c)}$ $\forall s^{(c)} \in C$, and $\boldsymbol{X}^{(q)}$ for a given query. Passing these through SetAggr_γ, we obtain $\boldsymbol{h}^{(q)}$, and a set of vectors $\{\boldsymbol{h}^{(c)}\}$.

Hyperplane Hashing We perform hyperplane hashing as follows: we choose K vectors $\boldsymbol{w}_i$ s.t. $\boldsymbol{w}_i \in \mathbb{R}^{d_{\text{aggr}}}$ $\forall i = [K]$, which may be identified as the normal vectors of corresponding hyperplanes passing through the origin. We denote a full set of K hyperplanes with the matrix $\boldsymbol{W} = [\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{K-1}]^\top \in \mathbb{R}^{K \times d_{\text{aggr}}}$. Using this, a K-bit hashcode for a given vector $\boldsymbol{h}$ may be generated as $\text{sign}(\boldsymbol{W}\boldsymbol{h}) \in \{\pm 1\}^K$, and the corpus set may be indexed into a hash table consisting of 2^K buckets.

We repeat the above process with L sets of hyperplanes $\widehat{W} = \{\boldsymbol{W}_i\}_{i=1}^L$ to generate L hash tables, and retrieve corpus sequences for a given q as:

$$\bigcup_{\boldsymbol{W} \in \widehat{W}} \{s^{(c)} : \text{sign}(\boldsymbol{W}\boldsymbol{h}^{(c)}) = \text{sign}(\boldsymbol{W}\boldsymbol{h}^{(q)})\}$$

Guarantees on Efficient LSH Retrieval LSH efficiency requires uniform bucketing, which in turn needs the map $\boldsymbol{X}^{(s)} \mapsto \boldsymbol{h}^{(s)}$ to be injective and *separable* w.r.t. a metric: if $\boldsymbol{X}^{(s)}, \boldsymbol{X}^{(t)}$ are well-separated (i.e., high negative $s_F(s,t)$), then $\boldsymbol{h}^{(s)}, \boldsymbol{h}^{(t)}$ must also be well separated in Euclidean space. We take SetAggr_γ to be permutation-invariant DeepSets (Zaheer et al., 2017), specifically, $G : \mathbb{R}^{|S| \times \dim_x} \to \mathbb{R}^{d_{\text{aggr}}}$, where,

$$G(\boldsymbol{X}^{(s)}) = \sum_{j=1}^{|S|} \text{ReLU}\Big(\boldsymbol{\Omega}\boldsymbol{X}^{(s)}[j] + \boldsymbol{\mu}\Big),$$

with trainable $\boldsymbol{\Omega} \in \mathbb{R}^{d_{\text{aggr}} \times \dim_x}$, $\boldsymbol{\mu} \in \mathbb{R}^{d_{\text{aggr}}}$ and pointwise ReLU.

Theorem 2. *Consider a query sequence $s^{(q)}$ and corpus $s^{(c)}$, with the corresponding embeddings $\boldsymbol{X}^{(q)}, \boldsymbol{X}^{(c)}$. We assume the embeddings are bounded, i.e $|\boldsymbol{X}_{ij}^{(q)}| \le B$ for some $B > 0$, and consider $\boldsymbol{\Omega}_j \overset{\text{iid}}{\sim} \mathcal{U}\left(\mathcal{S}^{d-1}\right), \boldsymbol{\mu}_j \overset{\text{iid}}{\sim} \mathcal{U}(-B, B)$, $\forall j \in [d_{aggr}]$. Then, there exists $c > 0$ such that*

$$\mathbb{E}_{\boldsymbol{\Omega},\boldsymbol{\mu}} \left\| G(\boldsymbol{X}^{(q)}) - G(\boldsymbol{X}^{(c)}) \right\|_2 \ge c \cdot \left\| \boldsymbol{X}^{(q)} - \boldsymbol{S}\boldsymbol{P}^*\boldsymbol{X}^{(c)} \right\|_F^2 \tag{10}$$

where G is the SetAggr_γ model defined above, and $\boldsymbol{P}^*$ is the optimal solution obtained from 4.

Proof. See Appendix F. □

Implications of Theorem 2 High negative $s_F(q,c)$ implies that $\boldsymbol{h}^{(q)}$ and $\boldsymbol{h}^{(c)}$ are well separated (in a Euclidean sense) under $G : \boldsymbol{X}^{(s)} \to \boldsymbol{h}^{(s)}$ for many $(\boldsymbol{\Omega}, \boldsymbol{\mu})$. For uniform bucketing, however, we need angular (and not just norm) separation, and a full DeepSets model with an MLP ρ_ϕ atop G can learn this. For two vectors on the hypersphere, angular and Euclidean separation align. For a bounded subset of $\mathbb{R}^d$, Appendix F.1 constructs a Bi-Hölder map into $\mathcal{S}^d$, and ρ_ϕ can approximate such a map, transferring Euclidean to angular bounds and yielding standard LSH guarantees.

Provable Uniform Bucketing For $V \subseteq \mathbb{R}^d$ with $\sup_{v \in V} \|v\| \le \psi$, then $\Phi(\boldsymbol{v}) := \left[\frac{\boldsymbol{v}}{\psi}, \sqrt{1 - (\frac{\|\boldsymbol{v}\|}{\psi})^2}\right]$ injects V into $\mathcal{S}^d$ and satisfies $\|\Phi(\boldsymbol{v}) - \Phi(\boldsymbol{u})\| \ge C\|v - u\|$ for some $C > 0$ depending only on d, ψ. Combined with Theorem 2, an MLP atop can emulate composition with a separable map to the hypersphere (e.g., Φ). Thus a DeepSet model $\text{MLP}(\sum_j \text{ReLU}(\boldsymbol{\Omega}\boldsymbol{X}^{(s)}[j] + \boldsymbol{b}))$ preserves distance and maps to a near-uniform hyperspheric region, where Euclidean and angular separations align, implying that dot-product hyperplane LSH preserves Euclidean separation on SetAggr_γ embeddings.

Using Neyshabur and Srebro (2015), we formalize:

Lemma 3. *For the optimal alignment $\boldsymbol{P}^*$ from 4, define $d(\boldsymbol{X}^{(q)}, \boldsymbol{X}^{(c)}) = \|\boldsymbol{X}^{(q)} - \boldsymbol{S}\boldsymbol{P}^*\boldsymbol{X}^{(c)}\|_F$.*
Then, for the asymmetric distance $d(\bullet, \bullet)$, there exists a (S, cS) ALSH using the SetAggr_γ embeddings.

Proof. See Appendix F.2. □

Practical Choice of Aggregator Given separable embeddings, we learn hyperplanes that account for the SetAggr_γ embedding distribution to ensure uniform bucketing. Although guarantees are shown for a specific SetAggr_γ, we use a Set Transformer (Lee et al., 2019) in practice for its superior performance and requiring minimal tuning, since even a tuned DeepSet lagged behind, in our experiments. Appendix H.4 provides empirical justification and shows similar performance profiles across the two architectures.

Random vs Trained Hyperplanes Choosing $\boldsymbol{w}_i$'s uniformly at random, i.e. $\boldsymbol{w}_i \sim \mathcal{N}(0, \mathbb{I})$ works well if the corpus items to be indexed are uniformly distributed in space. Such a uniform spatial distribution cannot be guaranteed for the embeddings generated through SetAggr_γ, and it inevitably leads to a poor load balance where most corpus items are indexed into a few buckets, leaving many others empty. These challenges motivate the use of *data driven* hyperplanes which divide corpus embeddings in a way that maintains desirable characteristics.

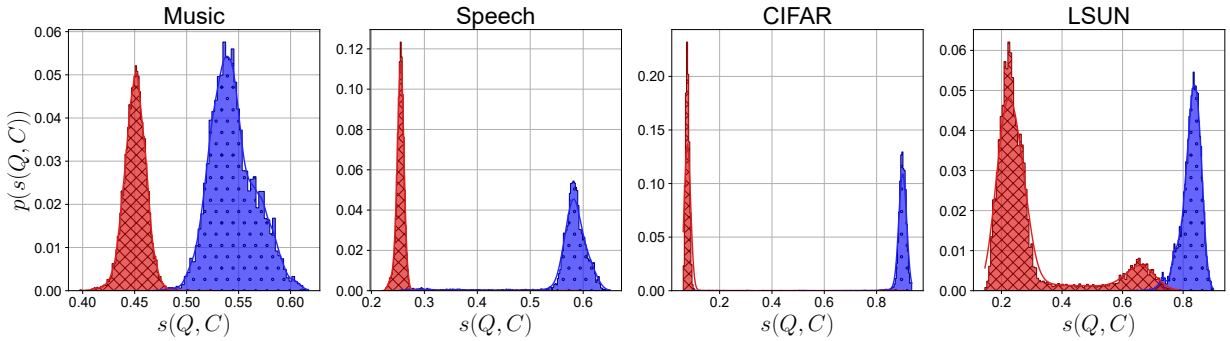

Figure 2: Empirical distribution of final score $s(q, c)$ for positive (Blue, dotted) and negative (Red, with cross hatches) query-corpus pairs in the test datasets (higher $s(q, c)$ signifies a better match).

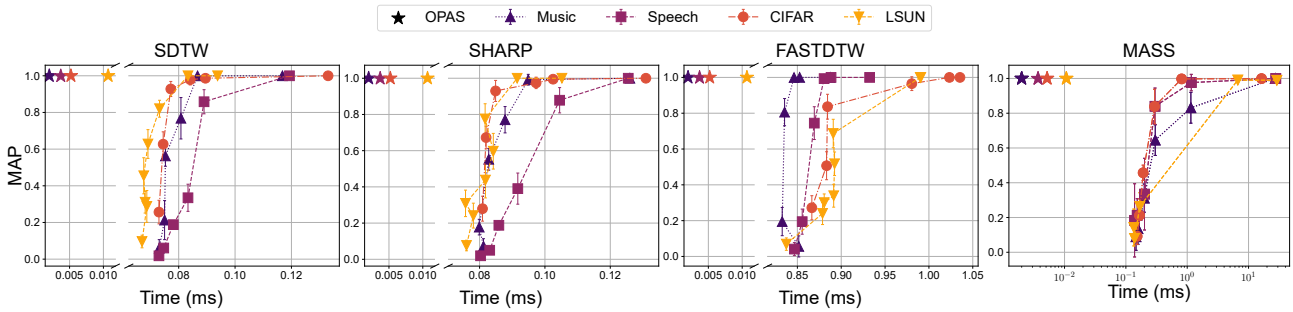

Figure 3: Retrieval performance (MAP) plotted vs the average time taken for a single query-corpus comparison. Points marked with ★ refer to OPAS, with different colors denoting the four datasets.

3.3.1 Training Details for Hyperplane Optimization

We replace sign(·) with tanh(·) as a smooth surrogate, and use a loss function with three key components– collision minimization, fence sitting, and bit balance– each improving it in a targeted manner. We modify the formulation introduced by Roy et al. (2023) by augmenting the bit balance loss defined as $\Delta_3' = \sum_{j\in[K]} \left|\sum_{s^{(c)}\in C} \tanh(\boldsymbol{W}\boldsymbol{h}^{(c)})[j]\right|$ using an additional regularizer. The original enforces even distribution of the set of corpus items in space by ensuring an equal number of $+1$'s and -1's at each index of the corpus hashcodes. Our proposed regularizer additionally accounts for *pairs* of $+1$'s and -1's within a given hashcode, ensuring that the pairs with product $+1$ (i.e., $(+1,+1)$ and $(-1,-1)$) occur with equal probability as pairs with product -1 (i.e. $(+1,-1)$ and $(-1,+1)$). We define the regularizer as:

$$R_3 = \sum_{j\in[K^2]} \left| \sum_{s^{(c)}\in C} (\tanh(\boldsymbol{W}\boldsymbol{h}^{(c)}) \otimes \tanh(\boldsymbol{W}\boldsymbol{h}^{(c)}))[j] \right| \tag{11}$$

The new bit balance loss is therefore defined, with hyperparameter α, as $\Delta_3 = \Delta_3' + \alpha R_3$. The final loss also includes a fence-sitting penalty and collision minimization component, for which we follow the original formulation. The exact choice of α, and the final loss used for training $\boldsymbol{W}$ are discussed in Appendix I.

4 EXPERIMENTS

In this section, we evaluate OPAS on the four datasets, analyze its key design components, and compare retrieval efficiency with a few metric-based baselines, and a popular transformer-based retrieval method ColBERT (Khattab and Zaharia, 2020).

4.1 Experimental Setup

Datasets We use two families of datasets—audio and image sequence. For audio, we extract audio from real-world videos from YouTube consisting of multilingual music (Music), and videos involving talks and presentations (Speech). Each clip is split into non-overlapping one-second windows of length d_{SR} (the audio sampling rate). A corpus sequence is formed using n contiguous windows, and the excess is discarded. For the image sequence datasets, we use CIFAR10 (Krizhevsky et al., 2009) and LSUN (churches subset) (Yu et al., 2015), denoted by CIFAR and LSUN. A corpus sequence here is constructed using one image augmented n times by randomized rotation, and queries are generated by sampling m frames uniformly from these. Query generation, splits, and labels are detailed in Appendix J. For retrieval, we follow a *split-corpus* setup, i.e., the corpus for all dataset splits are unique. As opposed to a fixed-corpus setup, this allows us to modify the test-time corpus size arbitrarily.

Experiment →	Default Parameters		Fixed $\lambda_j = 1\ \forall j$		$\beta_2 = 0$ in $s(q, c)$		$\boldsymbol{Z}' = \text{PHS}(\boldsymbol{X}^{(q)}, \boldsymbol{X}^{(c)})$	
Dataset ↓	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
Music	0.9934	0.9930	0.9761	0.9766	0.9781	0.9823	0.9931	0.9955
Speech	0.9935	0.9921	0.9248	0.9332	0.9179	0.9294	0.9869	0.9888
CIFAR	0.9993	1.0000	0.9967	0.9971	0.9471	0.9565	0.9766	0.9809
LSUN	0.9995	0.9996	0.8925	0.8963	0.7664	0.8759	0.7578	0.7920

Table 1: OPAS ablation studies verifying design choices for LAMMODEL_θ, SCORE_β, and cost matrix construction in Theorem 1.

Experiment →	Default Parameters		EMBED_ϕ arch: LRL		EMBED_ϕ arch: Conv1D		Pretraining EMBED_ϕ	
Dataset ↓	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
Music	0.9934	0.9930	0.9574	0.9541	0.9642	0.9594	0.9932	0.9923
Speech	0.9935	0.9921	0.9716	0.9701	0.9022	0.8963	0.9856	0.9850
CIFAR	0.9993	1.0000	0.5806	0.5206	0.2011	0.1426	0.9332	0.9533
LSUN	0.9995	0.9996	0.8242	0.8573	0.3063	0.2897	0.9517	0.9837

Table 2: OPAS ablations focusing on the embedding model EMBED_ϕ. Architectures and (self-supervised) pretraining details in Appendix L.11.

Evaluation Protocol Given a test query $s_{\text{test}}^{(q)}$, we score all items in C_{test}, compute average precision and reciprocal rank per query, and report MAP and MRR by averaging over test queries (Appendix N). To evaluate how well a scoring method performs in distinguishing ordered from unordered subsequence matches with corpus items, we define a new metric called *Order Discriminative Capacity* (ODC). This metric measures how well a scoring method prefers the correctly ordered subsequence over its shuffled variants. For a single query q, we create l shuffled versions and score all $l + 1$ queries with each positive corpus item of the original. We then compute the percentage of comparisons in which the original (ordered) query receives a higher score than each shuffled version, out of all comparisons. We define ODC (as a function of q, l, and $C_{q\checkmark}$) explicitly in Appendix N.1. Here, we fix $l = 6! - 1$, i.e., computing the ODC over all possible permutations for a sequence with $m = 6$.

4.2 Results

We summarize our key results in this section. Additional results are reported in Appendix L. Our code is available at: `https://github.com/BhavyaKohli/OPAS`

4.2.1 Comparing Retrieval Efficiency with Baselines

We compare OPAS to four metric-based baselines—FastDTW (Salvador and Chan, 2007), SoftDTW (Cuturi and Blondel, 2017), SHARP (Blondel et al., 2021), and MASS (Mueen et al., 2022). In Figure 3, we explore runtime–accuracy tradeoffs by reducing baseline complexity and windowing elements to $d' \ll d$. Even with reduced element size, pairwise-metric methods trail OPAS. On CPU, OPAS attains a speed up of at least 4.7X over the fastest baseline on Music (Table 5). We also recreate the audio datasets with $n = 500$ (compared to the original $n = 20$) and fixed $m = 6$ (Table 4) by reducing the effective audio sampling rate. Metric-based baselines deteriorate sharply as the corpus length increases, highlighting OPAS' scalability. For comparison with deep-learning baselines, we implement ColBERT on all datasets except LSUN (see Tables 6 and 7). See Appendix H.6 for more details on the implementation and hyperparameters used.

Method	MAP	MRR	Factor
$\text{OPAS}_{(\text{GPU})}$	0.5562	0.4885	1.00
SHARP	0.1145	0.0763	511.13
FASTDTW	0.2745	0.1893	990.31
SDTW	0.0380	0.0058	769.30
MASS	0.3040	0.0571	48,947.58

Table 4: Single query–corpus comparison on the long sequence Music dataset. Factor refers to the speed (averaged over 10 runs) relative to $\text{OPAS}_{(\text{GPU})}$, which has a single query-corpus comparison time of 0.0557 ms (results reported on a subset of the test set).

4.2.2 Analyzing Score Distributions for Positive and Negative Pairs

Figure 2 shows well-separated score distributions for positive vs negative pairs, supporting the implication of Theorem 2. In Appendix L.9, we also create a similar plot for $s_F(q, c)$, which follows a similar distribution, demonstrating how the embeddings themselves are well-separated, and not just the final scores.

Experiment →	Default Parameters		Hinge in $s_\lambda(q,c)$		Non-Sinkhorn SS_{exp}		Non-Sinkhorn SS_{sum}	
Dataset ↓	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
Music	0.9934	0.9930	0.9980	0.9982	0.8967	0.9087	0.9879	0.9884
Speech	0.9935	0.9921	0.9854	0.9843	0.9831	0.9821	0.9865	0.9863
CIFAR	0.9993	1.0000	0.9996	1.0000	0.9934	0.9947	0.9948	0.9961
LSUN	0.9995	0.9996	0.9983	0.9993	0.4606	0.4445	0.3301	0.3164

Table 3: Miscellaneous OPAS ablations. We introduce a hinge in $s_\lambda(q,c)$ to equalize scores where ordering is satisfied. Single-step (Non-Sinkhorn) variants in Appendix L.12.

Method	Time (ms)	Factor
$OPAS_{(GPU)}$	0.0020±0.0001	1.0
$OPAS_{(CPU)}$	0.0184±0.0009	9.2
SHARP	0.0867±0.0007	43.35
SDTW	0.0949±0.0014	47.45
FASTDTW	0.8462±0.0114	423.1
MASS	27.2378±0.0948	13618.9

Table 5: Single query–corpus comparison time on the Music dataset. Factor refers to the speed (averaged over 10 runs) relative to $OPAS_{(GPU)}$.

4.2.3 Comparing Retrieval Performance with ColBERT

In Table 6, we see that ColBERT performs well, although slightly worse than OPAS. Comparing their executing times on the Audio dataset, OPAS and ColBERT achieve similar retrieval efficiency, with single query-corpus comparison times of 1.9 μs and 2.5 μs respectively; and one-time corpus embedding times of 1.89 s and 1.64 s respectively, for 4,864 corpus items.

Method →	OPAS		ColBERT	
Dataset	MAP	MRR	MAP	MRR
Music	0.9934	0.9930	0.9853	0.9934
Speech	0.9935	0.9921	0.9417	0.9728
CIFAR	0.9993	1.0000	1.0000	1.0000

Table 6: ColBERT and retrieval performance on the two audio, and the CIFAR datasets.

Comparing the Order Discriminative Capacity OPAS and ColBERT are similar in terms of retrieval and execution speeds. However, ColBERT is designed to score full-document relevance, and uses max-similarity aggregation over embeddings in a bag-of-words fashion, not explicitly incorporating a mechanism for structured subsequence alignment into the score. As a result, it cannot explicitly distinguish between ordered and unordered occurrences of a query subsequence within a longer sequence. This is captured by the ODC metric, as seen in Table 7.

Dataset	OPAS	ColBERT
Music	$86.0509_{\pm 0.2126}$	$50.0048_{\pm 0.2407}$
Speech	$81.7590_{\pm 0.3988}$	$50.0054_{\pm 0.1630}$
CIFAR	$76.6300_{\pm 0.4496}$	$49.9850_{\pm 0.0401}$

Table 7: Comparison of OPAS and ColBERT on the ODC metric. Statistics reported over 10 runs.

4.2.4 Performance in LSH

We train hyperplanes (Section 3.3.1) and conduct *soft-indexing search*—items are retrieved from buckets with k common bits with $\boldsymbol{h}_q^{(b)}$. We vary k to study the trade off between the number of items retrieved and performance, and compare against FAISS (Johnson et al., 2019) indexes by matching the number of retrieved items in Figure 4, where we see our LSH pipeline showing a similar profile. Additionally, we note that the embeddings for the Music dataset favor inner products more than for Speech, which is likely a dataset-specific observation (See also Appendix L.13).

Implications Since LSH training is decoupled from OPAS, parity with FAISS indicates that the embeddings learned by OPAS are suitable for ANN search, thereby empirically supporting Theorem 2.

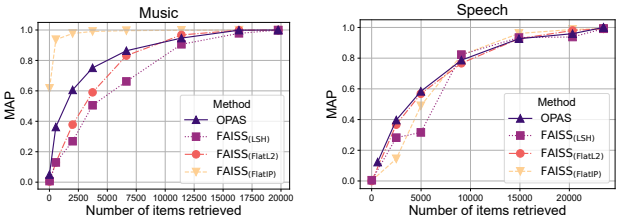

Figure 4: FAISS vs OPAS + LSH with trained hyperplanes.

4.2.5 Cross-Task Performance

We assess the cross-task generalization (Table 8) of OPAS by evaluating models trained on the audio datasets on each other, and similarly for models trained on the image sequence datasets. To perform this study, we ensure same embedding dimensionalities

so that models trained on one dataset may be applied directly on the test subset of the other. We also repeat this for the long sequence audio datasets in Table 9.

Source task	Eval on	MAP	MRR
Music	Speech	0.9852	0.9862
Speech	Music	0.9950	0.9949
CIFAR	LSUN	0.9461	0.9809
LSUN	CIFAR	0.9159	0.9242

Table 8: Cross-task performance on the audio and image sequence datasets.

Source task	Eval on	MAP	MRR
Music	Music	0.5076	0.4310
Music	Speech	0.2573	0.2024
Speech	Speech	0.8070	0.7721
Speech	Music	0.7267	0.6710

Table 9: Self and cross-task performance on the long sequence audio datasets.

4.2.6 Variable Length Corpus Sequences

Till now, we have considered datasets with a fixed sequence length n in the corpus, which may limit application to domains where such a strict structure is not possible. In Table 10, we experiment with this on the audio datasets; we recreate the two audio datasets with varying sequence lengths $n_i > m$ and $n_{\max} = 20$, similar to the original datasets. Similar to other sequence-related works, we simply pad corpus sequences with zeros to the same length $n_{\max}$. We note strong retrieval performance performance, which is expected since the dot-product similarity matrix $\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}$ automatically deals with padded sequences once we mask $\boldsymbol{X}^{(c)}$ using n_i, thereby only allowing the original sequence items to be matched and used for scoring.

Dataset	MAP	MRR
Music	0.9961	0.9964
Speech	0.9790	0.9794

Table 10: Retrieval performance of OPAS using varying-length corpus items

4.3 Ablation Studies

Table 1 tests three design choices in OPAS: (i) parameterizing $\boldsymbol{\lambda}$ (Column 2), (ii) incorporating the ordering criterion in $\textsc{Score}_\beta$ (Column 3), and (iii) the construction of $\boldsymbol{Z}$ in Theorem 1 (Column 4). For the alternative $\boldsymbol{Z}$, we use an asymmetric hinge-based reward matrix $\boldsymbol{Z}' = \mathrm{PHS}(\boldsymbol{X}^{(q)}, \boldsymbol{X}^{(c)})$ with $Z'[i][j] = -\sum(X^{(q)}[i] - X^{(c)}[j])_+$. These experiments show the need to: **(1)** parameterize $\boldsymbol{\lambda}$ to assign per-row importances; **(2)** include the ordering term in the scoring; and **(3)** exploit the task-specific reward matrix. All variants underperform the default design.

We ablate $\textsc{Embed}_\phi$ in Table 2. As stated in Section 3.1, we compare non-sequential models (LRL and Conv1D) that embed $\boldsymbol{x}_i^{(q)}, \boldsymbol{x}_j^{(c)} \to \mathbb{R}^{\dim_x}$ independently, observing a marked drop—supporting the choice to use a context-aware transformer for $\textsc{Embed}_\phi$. In another study, we pretrain $\textsc{Embed}_\phi$ in a self-supervised manner on a subset (by autoencoding noisy inputs) and then freeze it for training other OPAS components (Column 4). Performance drops only modestly, offering a practical alternative for users preferring a pretrain-and-freeze approach over end-to-end training.

Finally, Table 3 (Column 2) evaluates equalizing s_λ for positives by hinging $\boldsymbol{b} - \boldsymbol{APa}$, i.e., assigning the maximal ($= 0$) penalty once $\mathrm{diff}_{\boldsymbol{P}}(i) > b$. Columns 3–4 study *Single Step* normalization—$\mathrm{SS}_{\exp}$ and $\mathrm{SS}_{\mathrm{sum}}$—that row-normalize $\exp(\boldsymbol{Z})$ and $\boldsymbol{Z}$ respectively instead of running full Sinkhorn iterations (see Appendix L.12 for their explicit expressions). Although this reduces compute requirements, it degrades alignment quality and thus subsequence selection from $\boldsymbol{X}^{(c)}$, which is expected with $\boldsymbol{P}$ not being doubly stochastic.

5 CONCLUSION

In this work, we have presented OPAS, a novel neural framework for subsequence retrieval. By utilizing Sinkhorn iterations as a differentiable subsequence selection tool, we can approximate hard permutation matrices and efficiently use them to score hundreds of query-corpus pairs in parallel, allowing for quick retrieval of relevant items from a large corpus. Our experiments demonstrate the superiority of our framework compared to pairwise metric-based baselines, and demonstrate how the embeddings learned by OPAS are amenable to established ANN search methods such as LSH. Our ablation studies thoroughly stress each aspect of OPAS, and empirically validate our design choices through performance drops when they are not followed. With our comparison with ColBERT, a popular transformer-based retrieval method, we show how despite performing similarly in terms of retrieval metrics and efficiency, there are benefits of incorporating the ordering constraint into the scoring mechanism, which can be observed using the adversarial ODC metric. One limitation of applying OPAS to audio is the vulnerability of second-wise discretization to temporal distortions which directly affect the sequence length, which will require special care. Future work may focus more on the pre-embedding flow of this framework to improve its robustness to such distortions.

References

Adams, R. P. and Zemel, R. S. (2011). Ranking via sinkhorn propagation. *arXiv preprint arXiv:1106.1925*.

Baeza-Yates, R. A. (1991). Searching subsequences. *Theoretical Computer Science*, 78(2):363–376.

Berndt, D. J. and Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd international conference on knowledge discovery and data mining*, pages 359–370.

Blondel, M., Mensch, A., and Vert, J.-P. (2021). Differentiable divergences between time series. In *International Conference on Artificial Intelligence and Statistics*, pages 3853–3861. PMLR.

Boulnemour, I. and Boucheham, B. (2018). Qp-dtw: upgrading dynamic time warping to handle quasi periodic time series alignment. *Journal of Information Processing Systems*, 14(4):851–876.

Cao, D., Lin, Z., Liu, D., and Chai, X. (2025). G-wvdtw: A generalised weighted variance dynamic time warping algorithm for subsequence matching in multivariate time series. *Expert Systems*, 42(5):e70036.

Cuturi, M. (2013). Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26.

Cuturi, M. and Blondel, M. (2017). Soft-dtw: a differentiable loss function for time-series. In *International conference on machine learning*, pages 894–903. PMLR.

Daugelaite, J., O Driscoll, A., and Sleator, R. D. (2013). An overview of multiple sequence alignments and cloud computing in bioinformatics. *International Scholarly Research Notices*, 2013(1):615630.

Davidson, Y. and Dym, N. (2024). On the hölder stability of multiset and graph neural networks. *arXiv preprint arXiv:2406.06984*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

Han, S. and Zhang, L. (2022). Dynamic time warping under subsequence. In *4th International Conference on Information Science, Electrical, and Automation Engineering (ISEAE 2022)*, volume 12257, pages 514–519. SPIE.

Ibrahim, O., Hamed, B., and Abd El-Hafeez, T. (2022). A new fast technique for pattern matching in biological sequences. *The Journal of Supercomputing*, 79.

Johnson, J., Douze, M., and Jégou, H. (2019). Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

Khattab, O. and Zaharia, M. (2020). Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. (2019). Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR.

Liu, W., Mu, C., Kumar, S., and Chang, S.-F. (2014). Discrete graph hashing. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3419–3427, Cambridge, MA, USA. MIT Press.

Liu, W., Wang, J., Kumar, S., and Chang, S.-F. (2011). Hashing with graphs. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1–8.

Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization.

Mena, G., Belanger, D., Linderman, S., and Snoek, J. (2018). Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations*.

Montanari, P., Bartolini, I., Ciaccia, P., Patella, M., Ceri, S., and Masseroli, M. (2016). Pattern similarity search in genomic sequences. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):3053–3067.

Mueen, A., Zhong, S., Zhu, Y., Yeh, M., Kamgar, K., Viswanathan, K., Gupta, C., and Keogh, E. (2022). The fastest similarity search algorithm for time series subsequences under euclidean distance.

Neyshabur, B. and Srebro, N. (2015). On symmetric and asymmetric lshs for inner product search.

Papapetrou, P., Athitsos, V., Potamias, M., Kollios, G., and Gunopulos, D. (2011). Embedding-based

subsequence matching in time-series databases. *ACM Transactions on Database Systems (TODS)*, 36(3):1–39.

Park, S. and Chu, W. W. (2003). Similarity-based subsequence search in image sequence databases. *International Journal of Image and Graphics*, 3(01):31–53.

Pasternack, J. and Roth, D. (2009). Extracting article text from the web with maximum subsequence segmentation. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, page 971–980, New York, NY, USA. Association for Computing Machinery.

Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., and Keogh, E. (2012). Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270.

Roy, I., Agarwal, R., Chakrabarti, S., Dasgupta, A., and De, A. (2023). Locality sensitive hashing in fourier frequency domain for soft set containment search. *Advances in Neural Information Processing Systems*, 36:56352–56383.

Salvador, S. and Chan, P. (2007). Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580.

Shieh, J. and Keogh, E. (2008). i sax: indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631.

Sinkhorn, R. (1964). A relationship between arbitrary positive matrices and doubly stochastic matrices. *Annals of Mathematical Statistics*, 35:876–879.

Smith, T. and Waterman, M. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197.

Tahir, M., Sardaraz, M., and Ikram, A. A. (2017). Epma: Efficient pattern matching algorithm for dna sequences. *Expert Systems with Applications*, 80:162–170.

Van Do, L. and Anh, D. T. (2017). Time series motif discovery based on subsequence join under dynamic time warping. In *proceedings of the 2017 International Conference on Data Mining, Communications and Information Technology*, pages 1–5.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Yazdani, N. and Ozsoyoglu, Z. M. (1996). Sequence matching of images. In *Proceedings of 8th International Conference on Scientific and Statistical Data Base Management*, pages 53–62. IEEE.

Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., and Xiao, J. (2015). Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*.

Yun, C., Bhojanapalli, S., Rawat, A. S., Reddi, S. J., and Kumar, S. (2019). Are transformers universal approximators of sequence-to-sequence functions? *arXiv preprint arXiv:1912.10077*.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Zhao, J. and Itti, L. (2018). shapedtw: Shape dynamic time warping. *Pattern Recognition*, 74:171–184.

Zhou, M. and Wong, M. H. (2008). Efficient online subsequence searching in data streams under dynamic time warping distance. In *2008 IEEE 24th International Conference on Data Engineering*, pages 686–695. IEEE.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Not Applicable]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:

(a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes]

(b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]

(c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]

(d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:

   (a) Citations of the creator If your work uses existing assets. [Yes]

   (b) The license information of the assets, if applicable. [Not Applicable]

   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]

   (d) Information about consent from data providers/curators. [Not Applicable]

   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

   (a) The full text of instructions given to participants and screenshots. [Not Applicable]

   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]

   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Learning Right Monotone Permutation Matrices for Neural Subsequence Search: Supplementary Materials

A Limitations of our work

Our framework offers improvements in retrieval efficiency while performing at par with pairwise metric-based retrieval methods, and offers significant flexibility in data formats allowing for its use in virtually any domain where sequence elements can be embedded into vectors. However we point out some key limitations, including some already discussed in Section 5:

(1) Applying OPAS to audio is vulnerable to distortions such as compression and expansion, due to the second-wise discretization used. These distortions affect the sequence length and can therefore not be used in OPAS directly since all sequences are expected to be of uniform sizes $(m \times d), (n \times d)$. In Section 4.2.6, we see that variable-length sequences can be used, however this specific case is unique since the audio element itself changes with such distortions. Future work may focus on the pre-embedding flow of OPAS to improve its robustness to distortions including such as this.

(2) The increase in computational cost when it comes to longer query and corpus sequences, and the subsequent cost to compute Sinkhorn iterations and store gradients through them during training. This is a common drawback of methods involving Sinkhorn iterations. We have, however, demonstrated how this may be mitigated via our single-step ablation studies (Table 3, and Appendix L.12), which avoid expensive iterations at a reasonable loss in performance on most datasets.

B Broader impact

As mentioned earlier, we may apply the OPAS framework to subsequence based retrieval in virtually any domain, as long as the sequence elements can be embedded into vectors of reasonable quality. This may, for example, be extended to dynamic graphs–viewed as a sequence of graphs with individual graphs embedded into vectors using GNNs; human/robot actions–viewed as a sequence of a graph of coordinates and/or sensory data for limbs, with each state being embedded using a specialized model; and many more, as long as the application is sufficiently justified. Our design and the focus on using permutation matrices also adds a layer of interpretability to the retrieval mechanism, promoting transparency and reliability, as discussed and illustrated in Appendix L.15.

C LLM Usage

We used LLMs for minor editing and polishing of text, including fixing typos, grammatical errors, *etc.*

D Additional definitions and OPAS constants

D.1 Formal definitions of $\mathcal{P}_n$ and $\mathcal{B}_n$

$$\mathcal{P}_n = \left\{ \boldsymbol{P} \in \{0,1\}^{n \times n} \;\middle|\; \sum_{i=1}^{n} P_{ij} = \sum_{j=1}^{n} P_{ij} = 1 \right\} \qquad \mathcal{B}_n = \left\{ \boldsymbol{P} \in \mathbb{R}_{+}^{n \times n} \;\middle|\; \sum_{i=1}^{n} P_{ij} = \sum_{j=1}^{n} P_{ij} = 1 \right\} \tag{12}$$

D.2 Definition of A

$$\boldsymbol{A} = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 & 0 \\ -1 & 1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & -1 & 1 \end{pmatrix}_{m \times m} \tag{13}$$

D.3 Discussion on a and v

As defined in Section 3,

$$\boldsymbol{a} = \begin{bmatrix} v^0 & v^1 & \dots & v^{n-2} & v^{n-1} \end{bmatrix}^\top \in \mathbb{R}^n$$

Setting v to be too high causes numerical instability since exponentiation is involved (for $n = 20$, computing v^{19} can quickly get out of hand). In our experiments, we found that working with $v = 1.2$ gave a good tradeoff between imposing the ordering constraint using successive differences without being too high to cause numerical instability. In case n is very high, we instead use a simpler variant, as discussed in Appendix L.4.1.

E Proof of Theorem 1

For clarity, denote $\boldsymbol{X}^{(q)}$ by $\boldsymbol{Q}$, and $\boldsymbol{X}^{(c)}$ by $\boldsymbol{C}$, and define

$$\mathcal{T}(\boldsymbol{Q}, \boldsymbol{C}) := \|\boldsymbol{Q} - \boldsymbol{SPC}\|_F^2 + \langle \boldsymbol{\lambda}, \boldsymbol{b} - \boldsymbol{ASP}a \rangle \tag{14}$$

E.1 The First Term: $\|\boldsymbol{Q} - \boldsymbol{SPC}\|_F^2$

Consider the first term in the expression, and let $\hat{\boldsymbol{P}} := \boldsymbol{SP}$

$$\begin{aligned} \|\boldsymbol{Q} - \hat{\boldsymbol{P}}\boldsymbol{C}\|_F^2 &= \mathrm{tr}((\boldsymbol{Q} - \hat{\boldsymbol{P}}\boldsymbol{C})^\top (\boldsymbol{Q} - \hat{\boldsymbol{P}}\boldsymbol{C})) \\ &= \mathrm{tr}(\boldsymbol{Q}^\top \boldsymbol{Q}) - \mathrm{tr}(\boldsymbol{Q}^\top \hat{\boldsymbol{P}}\boldsymbol{C}) \\ &\quad - \mathrm{tr}(\boldsymbol{C}^\top \hat{\boldsymbol{P}}^\top \boldsymbol{Q}) + \mathrm{tr}(\boldsymbol{C}^\top \hat{\boldsymbol{P}}^\top \hat{\boldsymbol{P}}\boldsymbol{C}) \\ &= \mathrm{tr}(\boldsymbol{Q}^\top \boldsymbol{Q}) - 2\mathrm{tr}(\boldsymbol{Q}^\top \hat{\boldsymbol{P}}\boldsymbol{C}) \\ &\quad + \mathrm{tr}(\hat{\boldsymbol{P}}^\top \hat{\boldsymbol{P}}\boldsymbol{C}\boldsymbol{C}^\top) \end{aligned}$$

Note that

1. The $\mathrm{tr}(\boldsymbol{Q}^\top \boldsymbol{Q})$ term is a constant with respect to the permutation matrix $\boldsymbol{P}$
2. The second term can be written as:

$$\begin{aligned} \mathrm{tr}(\boldsymbol{Q}^\top (\hat{\boldsymbol{P}}\boldsymbol{C})) &= \sum_k \sum_i Q_{ik} \sum_j \hat{P}_{ij} C_{jk} \\ &= \sum_k \sum_i \sum_j Q_{ik} \hat{P}_{ij} C_{jk} \end{aligned}$$

3. The third term can be reduced as follows:

$$\text{Diagonal Elements of } \hat{\boldsymbol{P}}^\top \hat{\boldsymbol{P}} : \sum_i \hat{P}_{ij}$$

$$\text{Diagonal Elements of } \boldsymbol{C}\boldsymbol{C}^\top : \sum_j C_{ij}^2$$

Since $\hat{\boldsymbol{P}}^\top \hat{\boldsymbol{P}}$ is a diagonal matrix,

$$\begin{aligned}\operatorname{tr}(\hat{\boldsymbol{P}}^\top \hat{\boldsymbol{P}} \boldsymbol{C}\boldsymbol{C}^\top) &= \sum_j (\sum_i \hat{P}_{ij} \sum_k C_{jk}^2) \\ &= \sum_j \sum_i \sum_k \hat{P}_{ij} C_{jk}^2\end{aligned}$$

Using the above, we have

$$\begin{aligned}\|\boldsymbol{Q} - \hat{\boldsymbol{P}}\boldsymbol{C}\|_F^2 &= \operatorname{tr}(\boldsymbol{Q}^\top \boldsymbol{Q}) - 2\sum_k \sum_i \sum_j Q_{ik} \hat{P}_{ij} C_{jk} \\ &+ \sum_j \sum_i \sum_k \hat{P}_{ij} C_{jk}^2 \\ &= \operatorname{tr}(\boldsymbol{Q}^\top \boldsymbol{Q}) - \sum_i \sum_j \hat{P}_{ij} (2 \sum_k Q_{ik} C_{jk} - \sum_k C_{jk}^2)\end{aligned}$$

Note that

1. $\sum_k Q_{ik} C_{jk} = \sum_k Q_{ik} C_{kj}^\top = (\boldsymbol{Q}\boldsymbol{C}^\top)_{ij} = (\boldsymbol{C}\boldsymbol{Q}^\top)_{ji}$
2. $\sum_k \boldsymbol{C}_{jk}^2 = (\boldsymbol{C}\boldsymbol{C}^\top)_{jj}$

Define $\bar{\boldsymbol{C}} = \operatorname{diag}(\boldsymbol{C}\boldsymbol{C}^\top) \in \mathbb{R}^n$ and $\boldsymbol{Y} \in \mathbb{R}^{n \times m} := \bar{\boldsymbol{C}} \cdot \mathbb{1}_m^T$, and note that $(\boldsymbol{C}\boldsymbol{C}^\top)_{jj} = \bar{\boldsymbol{C}}_j = Y_{ji} \ \ \forall i$. Thus, we have

$$\begin{aligned}\|\boldsymbol{Q} - \hat{\boldsymbol{P}}\boldsymbol{C}\|_F^2 &= \operatorname{tr}(\boldsymbol{Q}^\top \boldsymbol{Q}) - \sum_i \sum_j \hat{P}_{ij} (2(\boldsymbol{C}\boldsymbol{Q}^\top)_{ji} - Y_{ji}) \\ &= \operatorname{tr}(\boldsymbol{Q}^\top \boldsymbol{Q}) - \sum_i \sum_j \hat{P}_{ij} (2\boldsymbol{C}\boldsymbol{Q}^\top - \boldsymbol{Y})_{ji} \\ &= \operatorname{tr}(\boldsymbol{Q}^\top \boldsymbol{Q}) - \operatorname{tr}(\hat{\boldsymbol{P}} \cdot (2\boldsymbol{C}\boldsymbol{Q}^\top - \boldsymbol{Y})) \\ &= \operatorname{tr}(\boldsymbol{Q}^\top \boldsymbol{Q}) - \operatorname{tr}(\boldsymbol{S}\boldsymbol{P} \cdot (2\boldsymbol{C}\boldsymbol{Q}^\top - \boldsymbol{Y})) \\ &= \operatorname{tr}(\boldsymbol{Q}^\top \boldsymbol{Q}) - \operatorname{tr}(\boldsymbol{P} \cdot (2\boldsymbol{C}\boldsymbol{Q}^\top - \boldsymbol{Y}) \cdot \boldsymbol{S}) \\ &= \operatorname{tr}(\boldsymbol{Q}^\top \boldsymbol{Q}) - \big(\operatorname{tr}(\boldsymbol{P} \cdot (2\boldsymbol{C}\boldsymbol{Q}^\top \boldsymbol{S})) - \operatorname{tr}(\boldsymbol{P}\boldsymbol{Y}\boldsymbol{S})\big)\end{aligned}$$

Using this, we have the following equivalent expression for finding the optimum:

$$\arg\min_{\boldsymbol{P} \in \mathcal{P}_N} \|\boldsymbol{Q} - \hat{\boldsymbol{P}}\boldsymbol{C}\|_F^2 = \arg\max_{\boldsymbol{P} \in \mathcal{P}_N} \left\langle \boldsymbol{P}, \big((2\boldsymbol{C}\boldsymbol{Q}^\top - \boldsymbol{Y}) \cdot \boldsymbol{S}\big)^\top \right\rangle_F \tag{15}$$

E.2 A special case

Since $\boldsymbol{Y} = \bar{\boldsymbol{C}} \cdot \mathbb{1}_m^\top$, we have $\operatorname{tr}(\boldsymbol{P}\boldsymbol{Y}\boldsymbol{S}) = \operatorname{tr}(\mathbb{1}_m^\top \boldsymbol{S}\boldsymbol{P}\bar{\boldsymbol{C}})$. If $\boldsymbol{C}$ is row-normalized to the same value c, i.e. $\sum_k C_{jk}^2 = c \ \forall k$, then all elements of $\bar{\boldsymbol{C}}$ will be identical ($= c$). In this case, $\boldsymbol{P}\bar{\boldsymbol{C}}$ will be independent of $\boldsymbol{P}$, and therefore, the following holds (the constant factor of 2 can be ignored):

$$\arg\min_{\boldsymbol{P} \in \mathcal{P}_N} \|\boldsymbol{Q} - \hat{\boldsymbol{P}}\boldsymbol{C}\|_F^2 = \arg\max_{\boldsymbol{P} \in \mathcal{P}_N} \langle \boldsymbol{P}, \boldsymbol{S}^\top \boldsymbol{Q}\boldsymbol{C}^\top \rangle_F \tag{16}$$

E.3 The Second Term: $\boldsymbol{\lambda}^\top(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{S}\boldsymbol{P}\boldsymbol{a})$

Consider the second term in the expression, and let $\hat{\boldsymbol{P}} := \boldsymbol{S}\boldsymbol{P}$

$$\boldsymbol{\lambda}^\top(\boldsymbol{A}\hat{\boldsymbol{P}}\boldsymbol{a} - \boldsymbol{b}) = \sum_k \lambda_k (\boldsymbol{A}\hat{\boldsymbol{P}}\boldsymbol{a} - \boldsymbol{b})_k$$

Now,

$$(\hat{\boldsymbol{P}}\boldsymbol{a})_i = \sum_j \hat{P}_{ij} a_j$$
$$(\boldsymbol{A}\hat{\boldsymbol{P}}\boldsymbol{a})_k = \sum_i A_{ki} (\hat{\boldsymbol{P}}\boldsymbol{a})_i = \sum_i A_{ki} \sum_j \hat{P}_{ij} a_j$$

Using $\sum_j \hat{P}_{ij} = 1 \ \forall i$, we can write

$$b = \sum_i \sum_j \hat{P}_{ij} \frac{b}{m}$$
$$\begin{aligned}(\boldsymbol{A}\hat{\boldsymbol{P}}\boldsymbol{a} - \boldsymbol{b})_k &= \sum_i A_{ki} \sum_j \hat{P}_{ij} a_j - \sum_i \sum_j \hat{P}_{ij} \frac{b}{m} \\ &= \sum_i \sum_j \hat{P}_{ij} \left(A_{ki} a_j - \frac{b}{m} \right)\end{aligned}$$

And therefore,

$$\boldsymbol{\lambda}^\top (\boldsymbol{A}\hat{\boldsymbol{P}}\boldsymbol{a} - \boldsymbol{b}) = \sum_i \sum_j \hat{P}_{ij} \left(\sum_k \lambda_k A_{ki} a_j - \sum_k \lambda_k \frac{b}{m} \right)$$

Construct matrices $\boldsymbol{\Gamma}$ and $\boldsymbol{W}$ as follows:

$$\boldsymbol{\Gamma} \in \mathbb{R}^{n \times m} := \left(\sum_k \lambda_k \right) \times (\mathbb{1}_n \cdot \mathbb{1}_m^\top)$$
$$\boldsymbol{W} \in \mathbb{R}^{n \times m} := \boldsymbol{a} \cdot \boldsymbol{\lambda}^\top \cdot \boldsymbol{A}$$

Note that the quantity $W_{ji} = a_j \cdot \sum_k \lambda_k A_{ki}$, and $\Gamma_{ji} = \sum_k \lambda_k$

Therefore we can write

$$\boldsymbol{\lambda}^\top (\boldsymbol{A}\hat{\boldsymbol{P}}\boldsymbol{a} - \boldsymbol{b}) = \sum_i \sum_j \hat{P}_{ij} \left(\boldsymbol{W} - \frac{b}{m}\boldsymbol{\Gamma} \right)_{ji} = \operatorname{tr}\left(\hat{\boldsymbol{P}} \cdot \left(\boldsymbol{W} - \frac{b}{m}\boldsymbol{\Gamma} \right) \right)$$

E.4 Combining the two terms

$$\begin{aligned}
&\arg\min_{\boldsymbol{P} \in \mathcal{P}_N} \mathcal{T}(\boldsymbol{Q}, \boldsymbol{C}) \\
=&\arg\min_{\boldsymbol{P} \in \mathcal{P}_N} \|\boldsymbol{Q} - \hat{\boldsymbol{P}}\boldsymbol{C}\|_F^2 - \boldsymbol{\lambda}^\top (\boldsymbol{A}\hat{\boldsymbol{P}}\boldsymbol{a} - \boldsymbol{b}) \\
=&\arg\min_{\boldsymbol{P} \in \mathcal{P}_N} -\operatorname{tr}(\hat{\boldsymbol{P}} \cdot (2\boldsymbol{C}\boldsymbol{Q}^\top - \boldsymbol{Y})) - \operatorname{tr}\left(\hat{\boldsymbol{P}} \cdot \left(\boldsymbol{W} - \frac{b}{m}\boldsymbol{\Gamma} \right) \right) \\
=&\arg\max_{\boldsymbol{P} \in \mathcal{P}_N} \operatorname{tr}(\hat{\boldsymbol{P}} \cdot (2\boldsymbol{C}\boldsymbol{Q}^\top - \boldsymbol{Y})) + tr\left(\hat{\boldsymbol{P}} \cdot \left(\boldsymbol{W} - \frac{b}{m}\boldsymbol{\Gamma} \right) \right) \\
=&\arg\max_{\boldsymbol{P} \in \mathcal{P}_N} \operatorname{tr}\left(\hat{\boldsymbol{P}} \cdot \left(2\boldsymbol{C}\boldsymbol{Q}^\top - \boldsymbol{Y} + \boldsymbol{W} - \frac{b}{m}\boldsymbol{\Gamma} \right) \right) \\
=&\arg\max_{\boldsymbol{P} \in \mathcal{P}_N} \operatorname{tr}\left(\boldsymbol{S}\boldsymbol{P} \cdot \left(2\boldsymbol{C}\boldsymbol{Q}^\top - \boldsymbol{Y} + \boldsymbol{W} - \frac{b}{m}\boldsymbol{\Gamma} \right) \right)
\end{aligned}$$

$$
\begin{aligned}
&= \arg\max_{\boldsymbol{P}\in\mathcal{P}_N} \operatorname{tr}\left(\boldsymbol{P}\cdot\left(2\boldsymbol{C}\boldsymbol{Q}^\top - \boldsymbol{Y} + \boldsymbol{W} - \frac{b}{m}\boldsymbol{\Gamma}\right)\cdot\boldsymbol{S}\right) \\
&= \arg\max_{\boldsymbol{P}\in\mathcal{P}_N} \operatorname{tr}(\boldsymbol{P}\cdot\boldsymbol{Z}^\top) = \arg\max_{\boldsymbol{P}\in\mathcal{P}_N} \langle \boldsymbol{P}, \boldsymbol{Z}\rangle_F
\end{aligned}
$$

We therefore have the final equivalent objective

$$
\arg\max_{\boldsymbol{P}\in\mathcal{P}_N} \langle \boldsymbol{P}, \boldsymbol{Z}\rangle_F = \arg\min_{\boldsymbol{P}\in\mathcal{P}_N} \|\boldsymbol{Q} - \hat{\boldsymbol{P}}\boldsymbol{C}\|_F^2 + \langle \boldsymbol{\lambda}, \boldsymbol{b} - \boldsymbol{A}\hat{\boldsymbol{P}}\boldsymbol{a}\rangle \tag{17}
$$

Where $\boldsymbol{Z} = \left(\left(2\boldsymbol{C}\boldsymbol{Q}^\top - \boldsymbol{Y} + \boldsymbol{W} - \frac{b}{m}\boldsymbol{\Gamma}\right)\cdot\boldsymbol{S}\right)^T \in \mathbb{R}^{n\times n}$

Reducing further $\boldsymbol{\Gamma}$ is a constant matrix, and will have no contribution in the optimization over $\boldsymbol{P}$. With the same structure on $\boldsymbol{C}$ as in Section E.2, we can additionally ignore $\boldsymbol{Y}$ in the expression for $\boldsymbol{Z}$ as well, giving us the reduced reward matrix:

$$
\begin{aligned}
\boldsymbol{Z} &= \left(\left(2\boldsymbol{C}\boldsymbol{Q}^\top + \boldsymbol{a}\cdot\boldsymbol{\lambda}^\top\cdot\boldsymbol{A}\right)\cdot\boldsymbol{S}\right)^T \\
&= \boldsymbol{S}^\top\left(2\boldsymbol{Q}\boldsymbol{C}^\top + \boldsymbol{A}^\top\boldsymbol{\lambda}\boldsymbol{a}^\top\right) \in \mathbb{R}^{n\times n}
\end{aligned} \tag{18}
$$

E.5 Using the reward matrix $\boldsymbol{Z}$

Since our optimization now has the form $\arg\max_{\boldsymbol{P}\in\mathcal{P}_N}\langle \boldsymbol{P}, \boldsymbol{Z}\rangle_F$, which is defined as the matching operator $M(\boldsymbol{Z})$, we may invoke Theorem 1 from Mena et al. (2018) and state the following, with $h(\boldsymbol{P}) = -\sum_{i,j} P_{i,j}\log(P_{i,j})$:

$$
\begin{aligned}
M(\boldsymbol{Z}) &= \lim_{\tau\to 0^+} \arg\max_{P\in\mathcal{B}_N} \langle \boldsymbol{P}, \boldsymbol{Z}\rangle_F + \tau h(\boldsymbol{P}) \\
&= \lim_{\tau\to 0^+} S(\boldsymbol{Z}/\tau)
\end{aligned} \tag{19}
$$

With this, we have converted our combinatorial search over $\mathcal{P}_N$ into a search for a soft permutation matrix in $\mathcal{B}_N$ which converges to the original optimum almost surely when $\tau \to 0$. In practice, very low values of τ lead to numerical instability, therefore we always work with the approximation when training.

This concludes the proof of Theorem 1.

F Proof of Theorem 2

Let $\mathbf{V}_1 = [x_1, \cdots, x_m]^\top \in \mathbb{R}^{m\times d}$ and $\mathbf{V}_2 = [y_1, \cdots, y_n] \in \mathbb{R}^{m\times d}$ be such that $m < n$ and $|x_i|, |y_i| < B$ for some $B > 0$ (this assumption is always satisfied if we row-normalize embeddings, see Appendix E.2). We define:

$$
\Delta(\mathbf{V}_1, \mathbf{V}_2) := \min_{\boldsymbol{P}\in\mathcal{P}_n} \sum_{i=1}^{m} \|\mathbf{V}_1[i] - (\boldsymbol{P}\mathbf{V}_2)[i]\|_2 \tag{20}
$$

Suppose $\boldsymbol{P}^*$ is the optimal permutation matrix that solves the minimization in $\Delta(\mathbf{V}_1, \mathbf{V}_2)$. Now, define:

$$
\tau : [m] \to [n] : \tau(i) := \text{the bit position of } \boldsymbol{P}^*[i] \tag{21}
$$

Then the coupling defined by $\tau(i)$, i.e., $(x_i, y_{\tau(i)})$, is the *optimal coupling* that minimizes the quantity $\min_{\kappa:[m]\to[n],\text{injective}} \left\|x_i - y_{\kappa[i]}\right\|$.

We consider $\boldsymbol{X}^{(q)}, \boldsymbol{X}^{(c)}$ as sets of vectors in $\mathbb{R}^{\dim_x}$, with the query sequence having m vectors and the corpus sequence having n vectors each. Following the trend of the task, we can safely assume $m < n$. We follow the notations from Davidson and Dym (2024), and consider the augmented Wasserstein distance $\mathcal{W}_n^T$, where we consider sets of size $\leq n$ and pad sets of size $< n$ with a vector $T \in \mathbb{R}^{\dim_x}$ where T lies outside of the domain of the query/corpus elements. Now we refer to Theorem 3.1 in (Davidson and Dym, 2024) where it is shown a

scalar set function of the form $G[d]$ (d^{th} output channel of the set-to-vector embedding G), with G as defined in Section 3.3, for sets of size $\leq n$ will satisfy the following lower bound:

$$\mathbb{E}_{\boldsymbol{\Omega}[d],\boldsymbol{\mu}[d]}\left|\sum_{i=1}^{m}\mathrm{ReLU}(\boldsymbol{\Omega}[d]^\top \boldsymbol{X}_i^{(q)}+\boldsymbol{\mu}[d]) - \sum_{j=1}^{n}\mathrm{ReLU}(\boldsymbol{\Omega}[d]^\top \boldsymbol{X}_j^{(q)}+\boldsymbol{\mu}[d])\right| \geq c\mathcal{W}_n^T(\boldsymbol{X}^{(q)},\boldsymbol{X}^{(c)})^2 \tag{22}$$

Now, as per the definition of $\Delta(\boldsymbol{X}^{(q)},\boldsymbol{X}^{(c)})$ from 20, and the subsequent definition of τ, we see that for any map $\mathcal{F}:[m]\to[n]$ must satisfy

$$\sum_{i=1}^{m}\left\|x_i - y_{\mathcal{F}(i)}\right\|_2 \geq \sum_{i=1}^{m}\left\|x_i - y_{\tau(i)}\right\|_2$$

Thus, if $\upsilon:[m]\to[n]$ is the map induced from the optimal coupling corresponding from the augmented Wasserstein $\mathcal{W}_n^T(\boldsymbol{X}^{(q)},\boldsymbol{X}^{(c)})$ then we must have that:

$$\begin{aligned}\mathcal{W}_n^T(\boldsymbol{X}^{(q)},\boldsymbol{X}^{(c)})^2 &\geq \sum_{i=1}^{m}\left\|x_i - y_{\upsilon(i)}\right\|_2^2\\ \geq \sum_{i=1}^{m}\left\|x_i - y_{\tau(i)}\right\|_2^2 &\geq \frac{1}{m}\left(\sum_{i=1}^{m}\left\|x_i - y_{\tau(i)}\right\|_2\right)^2\\ = \frac{\Delta(\boldsymbol{X}^{(q)},\boldsymbol{X}^{(c)})}{m}&\end{aligned}$$

Combining these, we get

$$\begin{aligned}&\mathbb{E}_{\boldsymbol{\Omega}[d],\boldsymbol{\mu}[d]}\left|\sum_{i=1}^{m}\mathrm{ReLU}(\boldsymbol{\Omega}[d]^\top \boldsymbol{X}_i^{(q)}+\boldsymbol{\mu}[d]) - \sum_{j=1}^{n}\mathrm{ReLU}(\boldsymbol{\Omega}[d]^\top \boldsymbol{X}_j^{(q)}+\boldsymbol{\mu}[d])\right|\\ &\geq C'\Delta(\boldsymbol{X}^{(q)},\boldsymbol{X}^{(c)})^2\end{aligned}$$

And by definition of Δ from 20, we see that

$$\begin{aligned}\Delta(\boldsymbol{X}^{(q)},\boldsymbol{X}^{(c)})^2 &\geq \sum_{i=1}^{m}\left\|\boldsymbol{X}^{(q)}[i] - (\boldsymbol{P}^*\boldsymbol{X}^{(c)})[i]\right\|_2^2\\ &= \left\|\boldsymbol{X}^{(q)} - \boldsymbol{S}\boldsymbol{P}^*\boldsymbol{X}^{(c)}\right\|_F^2\end{aligned}$$

Hence, we have

$$\begin{aligned}&\mathbb{E}_{\boldsymbol{\Omega},\boldsymbol{\mu}}\left\|\sum_{i=1}^{m}(\boldsymbol{\Omega}[d]^\top \boldsymbol{X}_i^{(q)}+\boldsymbol{\mu}[d])_+ - \sum_{j=1}^{n}(\boldsymbol{\Omega}[d]^\top \boldsymbol{X}_j^{(q)}+\boldsymbol{\mu}[d])_+\right\|_1\\ &= \sum_{d=1}^{d_{\text{aggr}}}\mathbb{E}_{\boldsymbol{\Omega}[d],\boldsymbol{\mu}[d]}\left|\sum_{i=1}^{m}\mathrm{ReLU}(\boldsymbol{\Omega}[d]^\top \boldsymbol{X}_i^{(q)}+\boldsymbol{\mu}[d]) - \sum_{j=1}^{n}\mathrm{ReLU}(\boldsymbol{\Omega}[d]^\top \boldsymbol{X}_j^{(q)}+\boldsymbol{\mu}[d])\right| \geq d_{\text{aggr}}C'\left\|\boldsymbol{X}^{(q)} - \boldsymbol{S}\boldsymbol{P}^*\boldsymbol{X}^{(c)}\right\|_F^2\end{aligned}$$

Note that, By Cauchy Schwarz on individual components of any vector $\boldsymbol{v}\in\mathbb{R}^d$, we get

$$\|\boldsymbol{v}\|_1 \leq \sqrt{d}\,\|\boldsymbol{v}\|_2$$

Thus, with the constant $C_2 := C'\sqrt{d_{\text{aggr}}}$ we obtain the bound:

$$\mathbb{E}_{\boldsymbol{\Omega},\boldsymbol{\mu}}\left\|\sum_{i=1}^{m}\mathrm{ReLU}(\boldsymbol{\Omega}^\top \boldsymbol{X}_i^{(q)}+\boldsymbol{\mu}) - \sum_{j=1}^{n}\mathrm{ReLU}(\boldsymbol{\Omega}^\top \boldsymbol{X}_j^{(q)}+\boldsymbol{\mu})\right\|_2 \geq C_2\left\|\boldsymbol{X}^{(q)} - \boldsymbol{S}\boldsymbol{P}^*\boldsymbol{X}^{(c)}\right\|_F^2$$

and our proof is complete.

F.1 Corollary and implications of Theorem 2

Provable Uniform Bucketing We now relate the guarantees from Theorem 2 with efficient LSH-based retrieval. For this, we first note the following: If $V \subseteq \mathbb{R}^d$ such that $\sup_{v \in V} \|v\| \leq \psi$, then the map $\Phi(\boldsymbol{v}) := \left[\frac{\boldsymbol{v}}{\psi}, \sqrt{1 - (\frac{\|\boldsymbol{v}\|}{\psi})^2}\right]$ takes V to $\mathcal{S}^d$ injectively and $\|\Phi(\boldsymbol{v}) - \Phi(\boldsymbol{u})\| \geq C \cdot \|v - u\|$ for some constant $C > 0$ depending only on d, ψ. Combining the above with the bounds from Theorem 2 we informally argue that, when a universal approximator such as an MLP is applied on top of G, then it can simulate composition with a *separable* map to the hyper-sphere, like Φ. Thus, a full DeepSet model like $\text{MLP}(\sum_j \text{ReLU}(\boldsymbol{\Omega}\boldsymbol{X}^{(s)}[j] + \boldsymbol{b}))$ can provably preserve the distance and map to a uniform hyper-spheric region. In this region, Euclidean separation and angular separation are analogous, which implies that the dot product based LSH using hyperplanes also preserve Euclidean separation when used for embeddings generated by SetAggr_γ.

We now use the results from Neyshabur and Srebro (2015) to formally show that, there exists an Asymmetric LSH on the SetAggr_γ embeddings

F.2 Proof of Lemma 3

Proof. From Theorem 5.3 of (Neyshabur and Srebro, 2015) we get that the inner product based hyperplane hashing gives an universal ALSH over $\left\{x \in \mathbb{R}^d : \|x\|_2 \leq 1\right\}$, which means hyperplane hashing is an (S, cS)-ALSH for every $S > 0, c \in (0, 1)$. Since the MLP_Θ outputs are norm bounded, this implies that SetAggr_γembeddings give an universal LSH on $\mathbb{R}^{d_{\text{aggr}}}$. It thus remains to show that the ALSH based on $L2$ distance on $\mathbb{R}^{d_{\text{aggr}}}$ also gives guarantees absed on $d(\boldsymbol{X}^{(q)}, \boldsymbol{X}^{(c)})$. Let $\zeta \in \mathbb{R}^d$ be the random vector normal to the hyperplane. Then we have:

$$
\begin{aligned}
&\mathbb{E}_{\zeta,\boldsymbol{\Omega},\boldsymbol{\mu},\Theta}\left|\zeta^\top \text{SetAggr}_\gamma(\boldsymbol{X}^{(q)}) - \zeta^\top \text{SetAggr}_\gamma(\boldsymbol{X}^{(c)})\right| \qquad !\\
&= \mathbb{E}_{\boldsymbol{\Omega},\boldsymbol{\mu},\Theta}\left[\mathbb{E}_\zeta\left[\zeta^\top(\text{SetAggr}_\gamma(\boldsymbol{X}^{(q)}) - \text{SetAggr}_\gamma(\boldsymbol{X}^{(c)}))\middle|\boldsymbol{\Omega}, \boldsymbol{\mu}, \Theta\right]\right. \qquad !\\
&\geq c\mathbb{E}_{\boldsymbol{\Omega},\boldsymbol{\mu}}\left[\left\|G(\boldsymbol{X}^{(q)}) - G(\boldsymbol{X}^{(c)})\right\|_2\right]\\
&\geq C'\left\|\boldsymbol{X}^{(q)} - \boldsymbol{S}\boldsymbol{P}^*\boldsymbol{X}^{(c)}\right\|_F^2 \text{ using Theorem 2}\\
&= C' d(\boldsymbol{X}^{(q)}, \boldsymbol{X}^{(c)})
\end{aligned}
$$

where $c, C' > 0$ are constants.

Thus, hyperplane hashing on SetAggr_γ is a universal ALSH on $d(\bullet, \bullet)$. □

Thus, we have guarantees on LSH using sequence embeddings and that is validated by our performance in retrieval as seen in our experiments.

G System Hardware

The system used for all GPU experiments is a compute server with an AMD EPYC 7742 processor and an NVIDIA A100-SXM4 GPU, using Python 3.8.18, PyTorch 2.2 and CUDA 11.8. Timing and memory profiling experiments for all baselines and $\text{OPAS}_{(\text{CPU})}$ were run locally on a laptop with an Intel(R) Core(TM) Ultra 7 155H processor and 32GB RAM. Typical GPU usage for training (using default batch sizes) on the audio datasets is 20-25GB (varies as PyTorch allocates and frees memory during the inference steps), and for the image sequence datasets is around 10-15GB (these are for the default batch sizes mentioned in Appendix H.1).

H Model architectures and hyperparameters

H.1 General hyperparameters

For LamModel_θ and Score_β we use the Adam optimizer (Kingma and Ba, 2014) with a fixed learning rate of 0.0005 and L_2 regularization parameter 10^{-5}, and for Embed_ϕ we use the AdamW optimizer (Loshchilov and Hutter, 2019) with weight decay 0.01 and a dynamic learning rate (linear warmup followed by decay) with the

maximum at 5×10^{-6} (Similar setup as in Devlin et al. (2019)). For training the image autoencoder PreEmbed_ω, we use the Adam optimizer with a learning rate of 0.001, with a StepLR scheduler to lower the learning rate by a factor of 0.8 when validation MSE does not improve over 5 epochs. We use a high batch size–400 for the Music, Speech and LSUN datasets, and 800 for CIFAR, in order to speed up training and encourage intra-batch diversity in query-corpus pairs. The margin Δ in the hinge loss defined in Equation 9 is set to 0.3 for the Music, Speech and LSUN datasets, and 0.7 for the CIFAR dataset. Most hyperparameters were taken from their respective method's original papers, and were not explicitly tuned. Grid-search type hyperparameter tuning was only performed for the LSH component (Table 11).

H.2 Embed_ϕ

We use a transformer encoder to embed a given input sequence (query or corpus). After adding a (fixed, cosine-based) positional encoding vector to the input sequence, the inputs are embedded using 2 standard transformer encoder layers (Vaswani et al., 2017) with a 4-way multi-headed attention module, and an input feedforward size of 512 for the audio datasets, and 256 for the image sequence datasets. The outputs of the transformer encoder are fed to a single linear layer, returning a final output of size ($l \times$ `out_dim`), where l is the length of the input sequence, and `out_dim` is 128 for the audio datasets, 32 and 64 for the CIFAR and LSUN image sequence datasets respectively.

H.3 LamModel_θ

LamModel_θ takes in the embedded query and corpus sequence in the form $\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top} \in \mathbb{R}^{m\times n}$. The architecture of the model is as follows: we use three sets of relu-activated convolutions followed by max-pooling (kernel size (1×2) for all three sets), with decreasing kernel sizes $(1\times n/4)$, $(1\times n/8)$, and $(1\times n/16)$ for the three sets. This is followed by a final 2D convolution layer with a m kernels of size $(1\times n/16)$, followed by adaptive pooling to size 20, netting a representation of shape $(m\times20)$. A ReLU-activated LRL network acts on this output, followed by a final sigmoid activation resulting in m values, one corresponding to each row of $\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}$. As discussed in Section 3.1, the first value is discarded since successive row differences are defined only from the second row.

H.4 SetAggr_γ

For our main SetAggr_γ, we use the official implementation of Set Transformer Lee et al. (2019) found at `https://github.com/juho-lee/set_transformer`. The architecture consists of two Induced Set Attention Blocks (ISAB) in the encoder, and a single Pooling by MultiHead Attention (PMA) module in the decoder.

Although not exactly rigorous, the trend (illustrated in Figure 5) expressed by both the DeepSet model and the Set Transformer are similar, and thus, we opted to use the Set Transformer for our experiments in the interest of performance and the fact that it did not need much hyperparameter tuning per-dataset as compared to the DeepSet model.

H.5 PreEmbed_ω for image sequence datasets

For both image sequence datasets, we train an autoencoder using the images using their original dataset splits. After training, we use the encoder to replace the $(C\times H\times W)$ sized images with their trained latent representation, which is flattened to a vector. This autoencoder training step is independent of OPAS training. Although we use an autoencoder architecture for PreEmbed_ω, it is not restrictive, and PreEmbed_ω can be trained in a supervised way (using image classes and/or other labels) as well. In our experiments, we trained OPAS using a ResNet-based classification model and a simple convolutional autoencoder on the CIFAR dataset (since class labels are required), and both gave similar performance. This shows that as long as the image embedding model is trained appropriately, the subsequent retrieval performance is maintained. Since the autoencoder is self-supervised, it is more generally applicable for an image dataset where labels might or might not exist (as is the case with LSUN, since we use images of only one class), and thus, we choose this as the image embedding model of choice.

The architectures of the autoencoder models for the two datasets are as follows: We use 7 (4) 2D convolution layers with a fixed kernel size of (4×4) with a stride of 2 for the LSUN (CIFAR) datasets, bringing the input of sizes $(3\times256\times256)$ and $(3\times32\times32)$ down to $(96\times2\times2)$ for both datasets. These are flattened when used

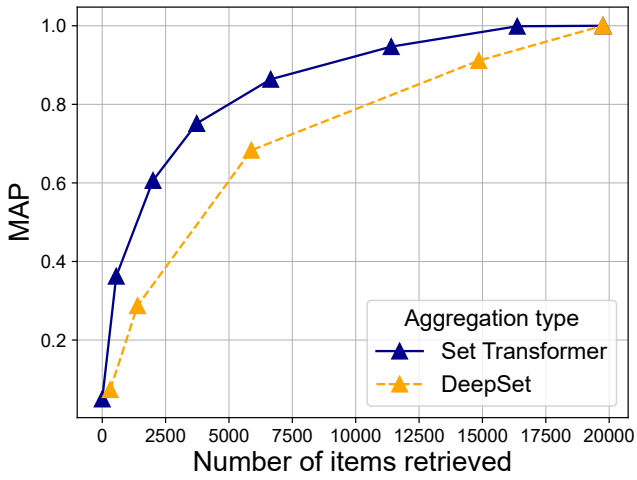

Figure 5: Set transformer vs DeepSet as $\textsc{SetAggr}_\gamma$, for the Music dataset

in OPAS to give a size 384 vector embedding. The vector embedding size for both image sequence datasets was kept identical on purpose, so that we may perform the cross-task experiments in Section 4.2.5.

H.6 ColBERT

For the ColBERT experiments we can apply the framework directly without the tokenization and token-embedding steps used for text, since our inputs are already vectors. In the original ColBERT setup, training uses $\langle q, d^+, d^- \rangle$ triplets to compute two scores and apply a cross-entropy loss. In our case, the training data contains relatively few positives per query, so we adopt the model architecture (with input-size changes appropriate for our setting) but train it with the hinge loss in Equation 9, using a higher margin $\Delta = 1$ (since the scores are not normalized to $[0, 1]$ as in OPAS and may be large in magnitude).

We add learned query and corpus identifier tokens, analogous to "[Q]" and "[D]", which make the effective query and corpus sequence lengths $m + 1$ and $n + 1$, respectively. Since all query and corpus sequences follow a fixed format, we do not need to handle attention masks for padding or filtering corpus items. Based on our experiments, we use a shallow configuration with 2 transformer layers, and choose the number of heads to be 10 for the audio datasets and 8 for CIFAR. Further experiments showed that deeper configurations tend to degrade performance in this setting. For the optimizer and learning rates, we use the default parameters suggested in Khattab and Zaharia (2020).

At test time, we use ColBERT's scoring function on the query and corpus embeddings to compute MAP and MRR. We do not implement the indexing component, and instead focus on the ranking capability of the learned embeddings using the defined score function.

I Training loss for learning hyperplanes

I.1 Collision minimizer

The collision minimizer loss is meant to drive hyperplane learning such that the bucket assigned to a given query q contains only positive items. We implement this using the hinge loss defined in Roy et al. (2023) as follows:

$$\Delta_1 = \sum_{s^{(q)} \in \mathcal{Q}} \sum_{s^{(c_+)} \in C'_{q\checkmark}, s^{(c_-)} \in C'_{q\times}} \left[1 + \tanh\left(\boldsymbol{W}\boldsymbol{h}^{(q)}\right)^\top \tanh\left(\boldsymbol{W}\boldsymbol{h}^{(c_-)}\right) - \tanh\left(\boldsymbol{W}\boldsymbol{h}^{(q)}\right)^\top \tanh\left(\boldsymbol{W}\boldsymbol{h}^{(c_+)}\right) \right] \tag{23}$$

As in Roy et al. (2023), the positive and negative pairs for this loss are identified by taking the top-k hashcode dot products as positives, i.e., *silver labels* and not the original gold ground truth labels y_{qc}.

I.2 Fence sitting

The fence sitting loss prevents the optimizer from arriving at the trivial solution $\boldsymbol{w}_i = 0$ by penalizing the L_1 distance of the generated hashcodes from 1. The loss is defined as follows:

$$\Delta_2 = \sum_{s^{(c)} \in C'} \| \tanh\left(\boldsymbol{W}\boldsymbol{h}^{(c)}\right) - 1 \|_1 \tag{24}$$

I.3 Bit balance

We discuss the bit balance loss in Section 3.3.1, and define it in Equation 11. The hyperparameter α on the additional regularizer is set to 0.001 for all datasets.

I.4 Final Loss

The final loss is a simple weighted sum $\sum_{i=1}^{3} \kappa_i \Delta_i$ of the three components $\Delta_{1,2,3}$, where κ_i are hyperparameters (See Table 11)

Dataset	κ_1	κ_2	κ_3
Audio	0.050	0.05	0.001
Speech	0.010	0.05	0.001
CIFAR	0.001	0.10	0.001

Table 11: Loss weight parameters for hyperplane training.

J Datasets

J.1 Audio

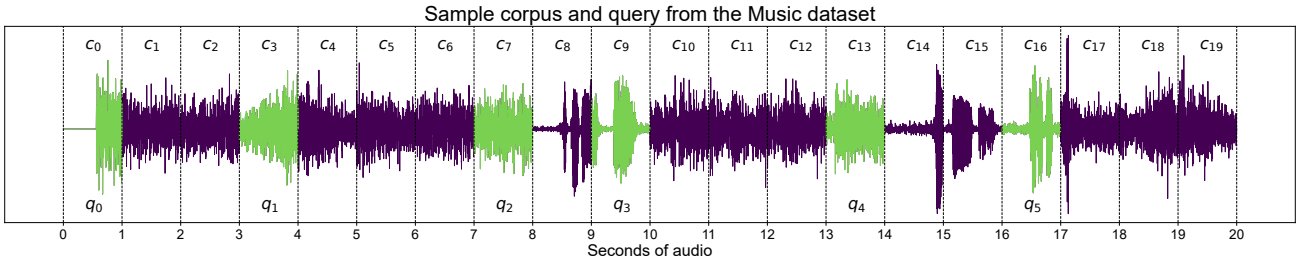

Figure 6: Sample corpus (violet) and a query generated from it (green). Individual seconds are demarcated by the vertical dotted lines, each second comprising of d_{SR} audio samples.

Both the audio datasets (Music and Speech) are multilingual and were downloaded using a curated set of YouTube playlists. Audio is first split into continuous windows of n seconds, and queries are generated by sampling m seconds uniformly at random (followed by sorting), giving us a single query-corpus pair. The corpus and query sequences are therefore of dim $(n \times d_{SR})$ and $(m \times d_{SR})$, where d_{SR} is the audio sampling rate.

To have multiple positive corpuses for a single query, we augment the original corpus sequence using zero masking at (multiple) random positions, simulating data loss and dropped data. Each query has a fixed number (16) of positive matches in the complete corpus sequence set after this step.

A sample corpus-query pair is visualized in Figure 6.

J.2 Image sequence

We work with two image datasets: (a) CIFAR10 and (b) LSUN churches (LSUN-C). CIFAR10 contains small (32×32) sized images, whereas LSUN-C contains larger images of size (256×256). Given one image, a single

sequence is generated by rotating the image from 0 to 270° in n steps. Queries are generated similarly to the audio dataset by sampling m frames UAR followed by sorting. Each rotation step is randomized, giving each query a fixed number (10) of positive matches in the complete corpus sequence set.

For use in OPAS, flattening images instead of learning representations directly will result in vectors of size 3072 and 196,608 respectively for CIFAR and LSUN, increasing memory requirements and execution times with the size of the input images. As discussed in Section 3.1, we use a pre-embedding network PreEmbed_ω to replace the images with their latent-space embeddings. OPAS therefore works with only the latent information in each image, which reduces memory requirements substantially. The corpus and query sequences before and after using a suitably trained embedding network are of size $(n \times C \times H \times W) \to (n \times d_{embed})$, and $(m \times C \times H \times W) \to (m \times d_{embed})$, where C, H, and W are the number of channels, height, and width of the images in their respective dataset, and d_{embed} is the latent dimension of the embedding network. More details about the image embedding models are discussed in Appendix H.5.

J.3 General data flow

Before each corpus item is fed into OPAS, it is further augmented with random noise with a fixed SNR. This is to ensure that the dot-product matrix $\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}$ does not contain exact matches when a given pair is positive.

J.4 Summary of dataset statistics

Dataset	For OPAS							For LSH	
	$\lvert Q_{\text{train}}\rvert$	$\lvert Q_{\text{val}}\rvert$	$\lvert Q_{\text{test}}\rvert$	$\lvert C_{\text{train}}\rvert$	$\lvert C_{\text{val}}\rvert$	$\lvert C_{\text{test}}\rvert$	$\lvert C_{q\checkmark}\rvert$	$\lvert C\rvert$	$\lvert C_{q\checkmark}\rvert$
Music	6000	3310	3040	9600	5296	4864	16	19760	16
Speech	3000	2195	2135	9600	7024	6832	16	23456	16
CIFAR	12000	12000	12000	6000	6000	6000	10	18000	10
LSUN	4000	2000	2000	2000	1000	1000	10	–	–

Table 12: Dataset statistics. The column "$\lvert C_{q\checkmark}\rvert$" represents the number of positive matches for a given query item in its respective split (and single global corpus set in the case of LSH) of corpus sequences.

Note that we **do not** perform LSH experiments on the LSUN dataset due to its track record of performance loss with increasing number of compression steps, and the LSH pipeline requires training two additional components—the SetAggr_γ, and the hyperplanes themselves. This is likely due to the high amount of base compression which PreEmbed_ω carries out, i.e. from $\mathbb{R}^{(3\times256\times256)}$ to $\mathbb{R}^{384}$, before the other models even get involved. Moreover, our aim with the LSH experiments is simply to show the amenability of the embeddings we learn with Embed_ϕ to LSH. Performance on the LSUN dataset may be improved by re-training a larger PreEmbed_ω from scratch, ensuring sufficient room for compression in later steps. As mentioned in H.5, we choose to maintain the same vector size for both CIFAR and LSUN for cross-task applicability in Section 4.2.5.

K Dataset Leakage Considerations

K.1 Audio

Assuming the dataset contains N_A audio files, we split we use $N_{A,tr}$ audio files (split into windows of n seconds) for training, and create validation and test subsets using the remaining $N_A - N_{A,tr}$ audio files. Validation and test sets are also kept similarly disjoint. Queries for all subsets are generated in the same way described in Appendix J.

K.2 Image Sequence

We ensure no train/val/test leakage similarly in this case. The image sequences are generated using images which were in the train/val/test splits used for training the autoencoders for both datasets. This automatically makes the three splits disjoint. For CIFAR, a set number of images are sampled from each class (in a list of classes) to ensure diversity in images. Since LSUN contains only one class of images, splits were generated by sampling UAR.

L Additional results

L.1 Simple LamModel$_\theta$

In Section 3.1, we discuss how LamModel$_\theta$ uses $\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}$ as the input to generate a specific $\boldsymbol{\lambda}$ for each query-corpus pair. In Table 13 (Column 2), we use a simpler architecture–a single linear layer (with bias) followed by sigmoid activation (to keep λ_j's positive). In contrast to the default architecture, this model takes a single concatenated sequence $[\boldsymbol{X}^{(q)}||\boldsymbol{X}^{(c)}]$ as input and is unable to utilize the pairwise similarity information encoded in $\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}$. As expected, this results in poor performance compared to the original setup.

Experiment →	Default Parameters		Simple LamModel$_\theta$		DeepSet$_1$+hinge		DeepSet$_2$+hinge	
Dataset ↓	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
Music	0.9934	0.9930	0.8660	0.8695	0.0281	0.0216	0.0415	0.0338
Speech	0.9935	0.9921	0.7939	0.8047	0.0231	0.0147	0.0225	0.0164
CIFAR	0.9993	1.0000	0.9744	0.9867	0.9741	0.9753	0.9341	0.9485
LSUN	0.9995	0.9996	0.1797	0.1464	0.4179	0.3449	0.2606	0.1998

Table 13: OPAS ablation studies with a focus on LamModel$_\theta$ and the scoring mechanism.

L.2 Scoring a query-corpus pair using their set embeddings

DeepSets (Zaheer et al., 2017) offer a way of embedding sequences of arbitrary length into a single unified representation. We leverage this by embedding a given query-corpus pair into vectors $\boldsymbol{h}_q$ and $\boldsymbol{h}_c \in \mathbb{R}^d$ which allow for straightforward comparisons using an asymmetric hinge-based score. In Table 13 (Columns 3-4), we define two scoring methods: (1) DeepSet$_1$+hinge, where we simply compute the un-normalized score $s(q,c) = -\sum(\boldsymbol{h}_q - \boldsymbol{h}_c)_+$; and (2) DeepSet$_2$+hinge, where we normalize this score to $[0,1]$ as $s(q,c) = 2\sigma\left(-\sum(\boldsymbol{h}_q - \boldsymbol{h}_c)_+\right)$. After computing $s(q,c)$, we train all models using the same margin loss defined in 9.

L.3 Stagger

We introduce a new concept called "Stagger", which aims at providing the $\boldsymbol{\lambda}$ generating neural network LamModel$_\theta$ additional information about the similarity information in $\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}$. By staggering inputs, we can allow a single convolution kernel to obtain similarity information about subsequent rows in $\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}$, thereby allowing it to generate $\boldsymbol{\lambda}$ such that the ordering constraint is better followed. This process is visualized (for stagger=2) in Figure 7.

In Table 14, we note no marked improvement in performance on the four datasets, however, in Table 15, we see that the additional information LamModel$_\theta$ has access to actually helps (in the case of the Music dataset) when we deal with more difficult datasets.

Stagger Value →	0 (default)		2	
Dataset ↓	MAP	MRR	MAP	MRR
Music	0.9934	0.9930	0.9936	0.9928
Speech	0.9935	0.9921	0.9854	0.9843
CIFAR	0.9993	1.0000	0.9996	0.9996
LSUN	0.9995	0.9996	0.9975	0.9988

Table 14: Effect of staggering inputs

L.4 Datasets with $n = 40$

We also recreate the four datasets but with longer corpus item lengths by setting $n = 40$ while keeping the query length fixed, i.e. $m = 6$. The results are in Table 15. The hyperparameters (negative exploration, $\dim_x$) slightly differ from the defaults for the audio datasets. The performance on the Music dataset degraded the

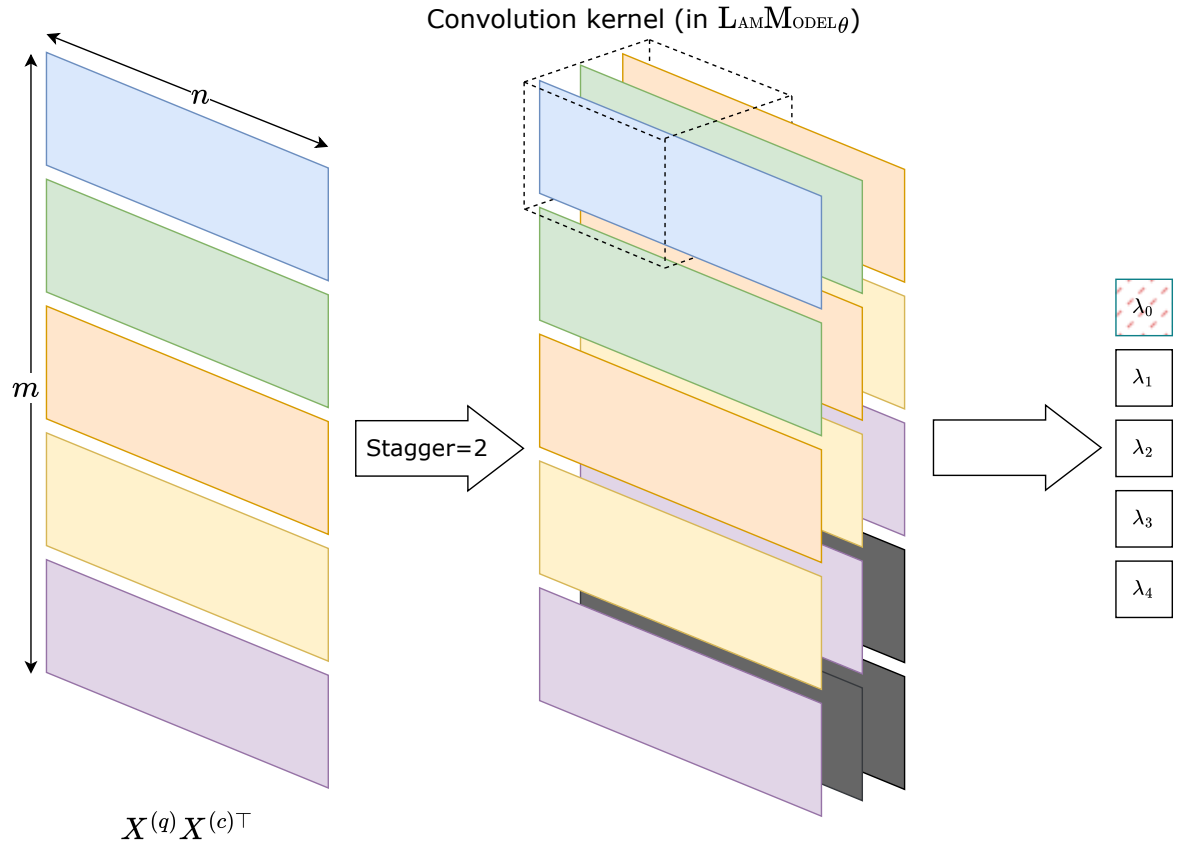

Figure 7: Visualizing the concept of staggering the input term $\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}$ in order to provide additional information to LamModel$_\theta$ when generating λ. In this illustrative example, $m = 5$. The dotted box represents a convolutional kernel within LamModel$_\theta$ which now acts not only on one row of $\boldsymbol{X}^{(q)}\boldsymbol{X}^{(c)\top}$, but multiple, giving it some information about later rows which may affect the ordering constraint. Dark gray boxes represent zero-padded vectors of size n

most, however, by setting stagger to 2, we managed to see improvements while keeping the other parameters constant.

Dataset	MAP	MRR
Music	0.9349	0.9309
Music (stagger=2)	0.9559	0.9556
Speech	0.9985	0.9976
CIFAR	1.0000	1.0000
LSUN	0.9993	0.9992

Table 15: Performance of OPAS on datasets with longer corpus sequence lengths

L.4.1 Regarding the 500-length dataset

When computing OPAS scores for the test dataset, if queries are to be scored with all corpus items simultaneously, the resulting permutation matrix tensor for per query causes GPUs to quickly go out of memory. To deal with this, corpus items are scored in smaller batches (increasing runtime as as result) and the scores concatenated together to get the final $(B \times |C_{test}|)$ score tensor, which is then used for computing MAP and MRR. The corpus item minibatch size has to be made even smaller for the $n = 500$ datasets, which is partly responsible for the

higher inference times (0.0020 ms to 0.0557 ms). Additionally the constant vector $\boldsymbol{a}$ is replaced with a simpler arange $\begin{bmatrix}0 & 1 & \dots & n-1\end{bmatrix}^\top$ since the exponentiation of v quickly goes out of hand when dealing with large n.

L.5 Sinkhorn sensitivity analysis

Sinkhorn iterations provide a valid permutation matrix only if the number of iterations are in the limit $l \to \infty$. As discussed in Section 3, we always work with the incomplete Sinkhorn operator, as an approximation. In Table 16, we study the effect of reducing the discretization of $\boldsymbol{P}$ by reducing the number of sinkhorn iterations.

Number of iterations →	20		15		10		5	
Dataset ↓	MAP	MRR	MAP	MRR	MAP	MRR	MAP	MRR
Music	0.9934	0.9930	0.9896	0.9929	0.9880	0.9880	0.9936	0.9942
Speech	0.9935	0.9921	0.9840	0.9837	0.9773	0.9767	0.9771	0.9759
CIFAR	0.9993	1.0000	0.9980	0.9981	0.9999	1.0000	0.9994	1.0000
LSUN	0.9995	0.9996	0.8822	0.8899	0.8792	0.8863	0.9053	0.9106

Table 16: Effect of the number of Sinkhorn Iterations on the method's performance

L.6 Simpler Score$_\beta$ with no parameters

In Table 17, we see our original results on the four datasets without SCORE$_\beta$, where the final score is simply given by $\sigma(s_F + s_\lambda)$. This also follows a similar trend as the other experiments, speech being hit more than music in the audio datasets, CIFAR not being affected because of its apparent ease, and LSUN being negatively affected. Since adding two scalar parameters β_1 and β_2 does not significantly add to the computational overhead, we simple included them in the base version of OPAS.

Dataset	MAP	MRR
Music	0.9617	0.9674
Speech	0.8811	0.8921
CIFAR	0.9998	1
LSUN	0.9558	0.9627

Table 17: Performance of OPAS without a parameterized scoring function.

L.7 Using a powerful contextual embedding model with the same objective as Section 3

In this experiment, we use an alternative approach, using a concatenated sequence $q||c$ of length $m+n$ as the input to a transformer. The output of this, say, $\boldsymbol{X}_{qc}$ of shape $((m+n) \times d)$, is passed through a readout layer resulting in a single score for this specific pair. Since each query needs to be individually concatenated with all corpus items separately, this slows down the evaluation process, and consequently the training times, considerably.

Dataset	MAP	MRR
Music	0.0143	0.0076
Speech	0.0149	0.0089
CIFAR	0.2592	0.2077
LSUN	0.7951	0.7940

Table 18: Performance of OPAS with direct scoring using a powerful embedding model EMBED$_\phi$.

These results follow a loosely similar trend to the DeepSets experiment in Appendix L.2, where the audio datasets are affected significantly while the image sequence datasets are not directly zero-ed out. With some more tuning

it may be possible to improve performance on the image sequence datasets to a similar level as the DeepSets experiment, however, at the cost of much higher training time due to slow evaluations.

Although there are merits to this approach, being much simpler and easier to implement, it puts a lot of work on the embedding model, meaning it is by nature more susceptible to hyperparameter-based issues.

L.8 Sparse sampling of negative Q-C pairs (negative exploration)

Given the dataset construction, we have $|Q| = M$ queries and $|C| = N$ corpus items, with each query $s^{(q)}$ having a set number of positive matches $C_{q\checkmark}$ and negative matches $C_{q\boldsymbol{\times}} = C \setminus C_{q\checkmark}$. Using all possible pairs would lead to a severe imbalance of positives and negatives, which may cause a bias. To partly alleviate this, we sample a fixed number of negative corpus items for each query ("negative exploration"), which additionally reduces training times as a side effect. In this study, we observe the effects of changing the amount of data available to OPAS models during training.

Negative Exploration →	800 (default)		400		200	
Dataset ↓	MAP	MRR	MAP	MRR	MAP	MRR
Music	0.9934	0.9930	0.9899	0.9917	0.9835	0.9854
Speech	0.9935	0.9921	0.9840	0.9837	0.9326	0.9357
CIFAR	0.9993	1.0000	0.9976	0.9970	0.9984	0.9986
LSUN	0.9995	0.9996	0.7860	0.7923	0.7909	0.7928

Table 19: Effect of sampling a fixed number of negative query-corpus pairs during OPAS training. Negative pairs for query $s^{(q)}$ are created by pairing it with corpus items in $C_{q\boldsymbol{\times}}$ uniformly at random

L.9 Visualizing score distributions for positive and negative query-corpus pairs

Similar to Figure 2, we plot the empirical distribution of the s_F component of $\textsc{Score}_\beta$, and note the same observation: The alignments learned by OPAS result in a well separated $s_F(q, c)$.

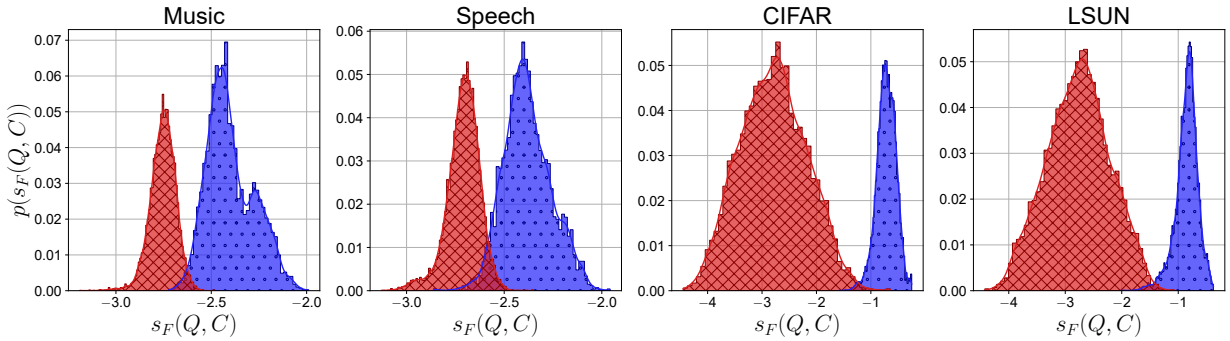

Figure 8: Empirical distribution of norm component $s_F(q, c)$ for positive (Blue, dotted) and negative (Red, with cross hatches) query-corpus pairs (higher $s_F(q, c)$ signifies a better match) for the test subset of the four datasets.

L.10 Visualizing the quality of learned matrices vs predicted scores for the positive query-corpus pairs

In Figures 9 and 10, we visualize how the distance between learned alignments from the gold alignment varies with the final assigned scores $s(q, c)$ and $s_F(q, c)$. Intuitively, the gap $\|\boldsymbol{P} - \boldsymbol{P}^*\|_F^2$ should *decrease* as score increases, since a smaller value of this norm would signify the alignment returned by OPAS closely matches the ground truth. This trend of lower norm difference (learned $\boldsymbol{P}$ is closer to the ground truth) corresponding to a higher $s(q, c)$ (and $s_F(q, c)$), however, is more apparent in the audio datasets.

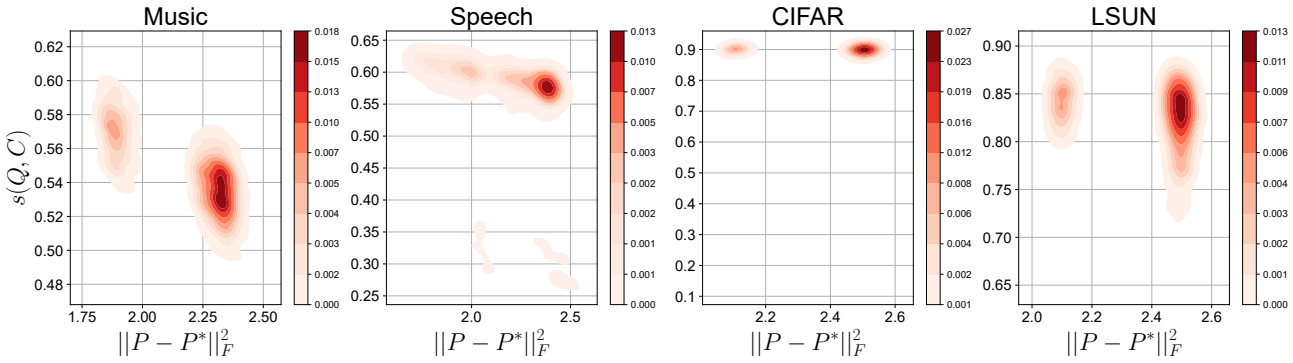

Figure 9: Norm difference of the learned alignment and the ground truth, plotted vs the final score $s(q, c)$ (higher $s(q, c)$ signifies a better match)

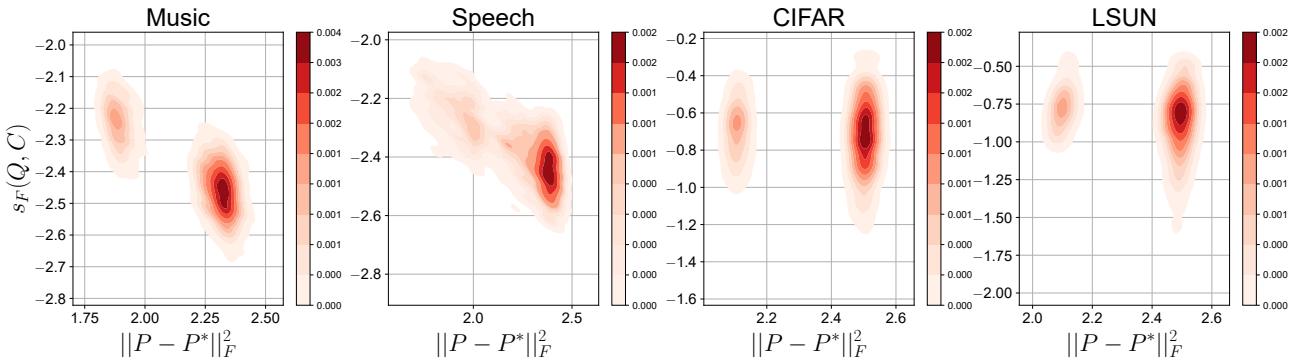

Figure 10: Norm difference of the learned alignment and the ground truth, plotted vs $s_F(q, c)$ (higher $s_F(q, c)$ signifies a better match)

L.11 Testing different architectures for Embed$_\phi$

As described in Section 4.3, in the experiments pertaining to Table 2, we focus on EMBED$_\phi$ and design choices associated with it. In particular, we use two non-sequential architectures–LRL and Conv1D. The LRL model is a simple Linear-ReLU-Linear model, and the Conv1D model carries out three rounds of grouped operations–1D convolutions followed by batch normalization and max-pooling, followed by adaptive pooling to size 1024. The convolution layers have an increasing number of filters, increasing from 1 (input) to 4, 8, and finally 16 over the three convolution sets. The output of these three sets followed by adaptive pooling are fed into an LRL network which embeds them into $\mathbb{R}^{\text{dim}_x}$.

In both these cases, the models used act on individual sequence elements and do not benefit from contextual information unlike the transformer, which reduces their embedding quality.

L.12 Single step normalization

As described in Section 4.3, in the experiments pertaining to Single Step normalization in Table 2, we define the operations SS_{exp} and SS_{sum} on a given reward matrix $\boldsymbol{Z}$ as transformations which return $\boldsymbol{Z}'_{\text{exp}}$ and $\boldsymbol{Z}'_{\text{sum}}$ respectively, where $\boldsymbol{Z}'_{\text{exp}}$ and $\boldsymbol{Z}'_{\text{sum}}$ are defined as follows:

$$\boldsymbol{Z}'_{\text{exp}}[i][k] = \frac{\exp(Z[i][k])}{\sum_{j=0}^{n-1} \exp(Z[i][j])} \tag{25}$$

$$\boldsymbol{Z}'_{\text{sum}}[i][k] = \frac{Z[i][k]}{\sum_{j=0}^{n-1} Z[i][j]} \tag{26}$$

These steps serve as a single step alternative to Sinkhorn iterations. They row-normalize the reward matrix $\boldsymbol{Z}$ in an attempt to assign importances (summing to 1) to each row in $\boldsymbol{X}^{(c)}$, similar to how the attention mechanism in transformers assigns an importance value or weight to tokens in the input stream.

L.13 Retrieval performance using mbedding-space metrics on SetAggr$_\gamma$ trained embeddings

After training SetAggr$_\gamma$ we compare retrieval performance using the original OPAS scores, vs distance metrics in the embedding vector space. Here, IP refers to the inner product similarity $IP(q,c) = \langle \boldsymbol{h}_q, \boldsymbol{h}_c \rangle$ and CS refers to the cosine similarity $CS(q,c) = \frac{IP(q,c)}{\|\boldsymbol{h}_q\| \cdot \|\boldsymbol{h}_c\|}$. Significant differences across the two datasets may be attributed to the large amount of transformation and compression which is being done on the original sequence as it is passed through Embed$_\phi$, LamModel$_\theta$, and finally SetAggr$_\gamma$. To verify the discrepency as not a one-off event, we run SetAggr$_\gamma$ multiple times with the same and slightly different parameters and obtain MAP $= 0.0048 \pm 0.0015$, MRR $= 0.0026 \pm 0.0009$ (The numbers in the table is the performance of the overall best-performing SetAggr$_\gamma$).

Dataset →	Music		Speech	
Method ↓	MAP	MRR	MAP	MRR
OPAS	0.9934	0.9930	0.9935	0.9921
IP	0.5824	0.5952	0.0038	0.0022
CS	0.5836	0.6001	0.5676	0.5867

Table 20: Non-FAISS metrics in embedding space on query-corpus embeddings $\boldsymbol{h}_q$ and $\boldsymbol{h}_c$ after training SetAggr$_\gamma$

L.14 LSH on CIFAR

Over all our experiments, it is clear that CIFAR is a relatively easy dataset, with it being more robust to design changes than the other datasets in most cases. In Table 21, we observe that even with the number of common bits (k_{com}) set to the maximum, effectively retrieving just one bucket from each hash table, we obtain a MAP of 1.

k_{com}	MAP	Avg. num retrieved
10	1.0000	947.11
8	1.0000	2297.26
6	1.0000	6637.90
4	1.0000	14822.15
2	1.0000	17995.44
1	1.0000	18000.00

Table 21: Performance of LSH with trained hyperplanes on CIFAR (D3)

L.15 Visualizing learned alignments

After training the OPAS models, we visualize the alignments $\boldsymbol{P}$ which are internally being used for selecting subsequences from a specific corpus item. The learned alignments for a few query-corpus pairs in the music dataset are illustrated in Figure 11. A lower τ gets us a more accurate approximation of the hard permutation matrix $\boldsymbol{P}^*$ which maximizes $\langle \boldsymbol{P}, \boldsymbol{Z} \rangle$. During training, however, we prefer to work with a higher τ, and consequently a *softer* approximation $\boldsymbol{P}^*$ because a lower τ causes unstable gradients. During inference, however, to obtain hard permutation matrices, we can either opt to use `scipy`'s implementation of linear sum assignment directly (on matrix $-\boldsymbol{Z}$), or a low τ to get a more discretized $\boldsymbol{P}$, since there are no gradients to compute. In Figure 11, we observe just that–a lower τ approximates the results of the linear sum assignment much more closely, and the resulting alignment matches the ordering used to generate the queries using the respective corpus items. In practice, since our objective is to use the final scores and not work with the alignments themselves, we do not require this high level of discretization and use the same τ during inference.

`https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linear_sum_assignment.html`

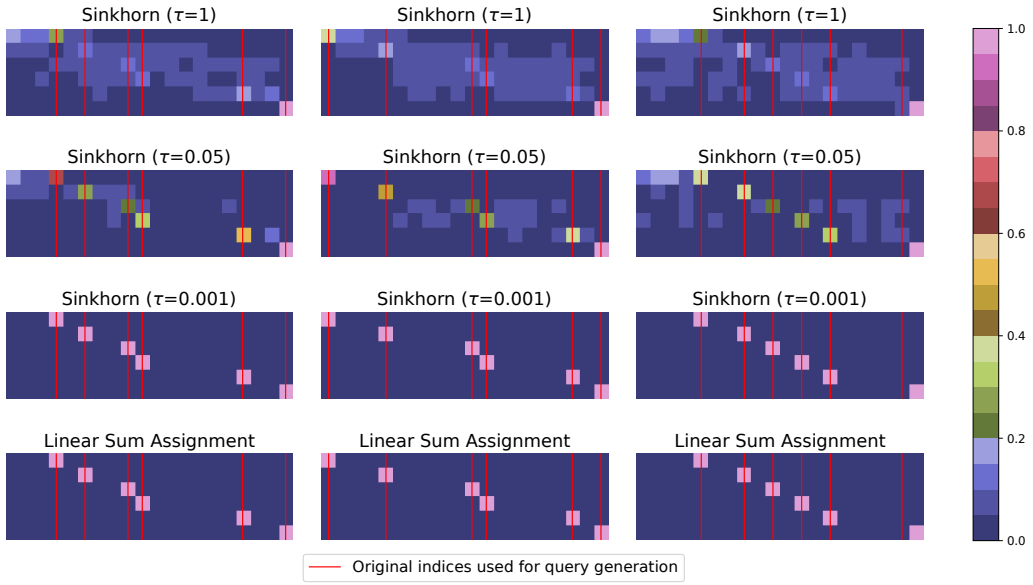

Figure 11: Visualizing the internal learned alignment for a few query-corpus pairs

L.16 MAP vs memory, baselines

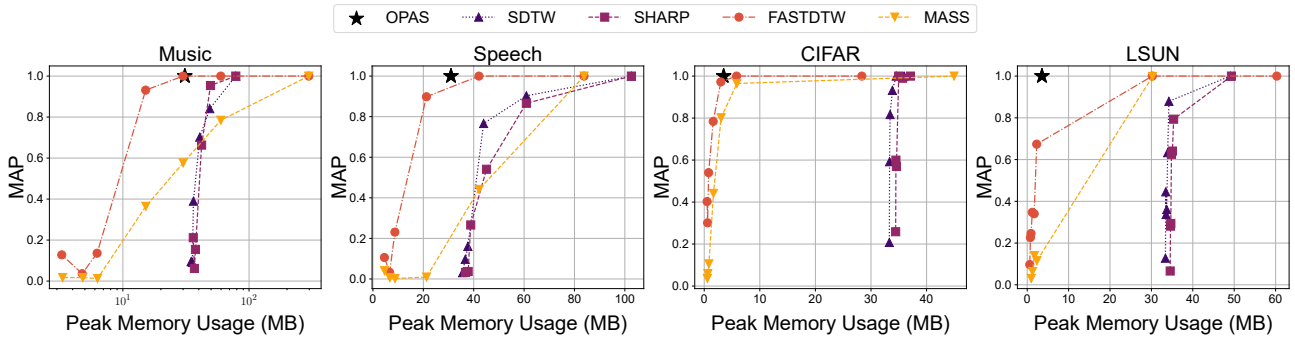

Figure 12: MAP vs Peak memory usage, as measured using tracemalloc

In a controlled environment, the maximum allocated memory as measured using `tracemalloc`, for OPAS and all baselines is shown in Figure 12. From each dataset, a set number B of queries were picked (at random), and each method compares these queries to the complete corpus set. Maximum allocated memory is computed over the complete duration of inference, i.e., computing BN scores for each dataset, for each method. Compared to the baselines on every dataset, OPAS has lower peak memory utilization at peak performance (MAP ≈ 1).

M Obtaining Timing and Memory Data

M.1 Timing data

All timing experiments were run locally on a laptop (Appendix G) with a very constrained setup to ensure low inter-run variance in measurements. The conditions which were used for obtaining all data used in our figures and tables are:

1. No two experiments were run parallelly: Each data point in the timing experiment graph was obtained by running the script (with its corresponding arguments) with no other instance of the script at the time.

`https://docs.python.org/3/library/tracemalloc.html`

2. No open applications other than the terminal: Apart from the terminal software (Microsoft *Terminal Preview*) running Powershell 7.4.5, no other window application was running.

M.2 Memory data

For memory profiling, we chose to use *tracemalloc* among other libraries, since it gave the most consistent results across multiple runs. For avoiding any possibility of a memory leak, we maintain the same conditions for obtaining this data as described above.

N Evaluation metrics

For a given batch of queries $\mathcal{Q}$ and corpus set C, we first define the following quantities, where *rank* refers to the index (starting from 1) of corpus items sorted in decreasing order using scores $s(q, c)\ \forall c \in C$:

1. r_q as the rank of the first relevant corpus item for query q.
2. $\text{rel}_q(k)$, which is 1 if the corpus item at rank k is relevant to query q, and 0 otherwise.
3. TR_q as the total number of relevant corpus items for query q.
4. $\text{P}_q(k)$ as Precision@k, which is the fraction of relevant items among the top k ranked corpus items.

Using these, we have:

$$\text{MRR}_q = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \frac{1}{r_q} \tag{27}$$

$$\text{AP}_q = \frac{\sum_{k=1}^{|C|} \text{P}_q(k)\ \text{rel}_q(k)}{\text{TR}_q} \tag{28}$$

$$\text{MAP} = \frac{1}{|\mathcal{Q}|} \sum_{q \in \mathcal{Q}} \text{AP}_q \tag{29}$$

N.1 Order-Discriminative Capacity

With $\text{Shf}(q, l)$ as the set of l copies of a given query q, each shuffled along the time axis, we define the ODC as:

$$\text{ODC}(q, l) = \frac{1}{l \cdot |C_{q\checkmark}|} \sum_{c \in C_{q\checkmark}} \sum_{\tilde{q} \in \text{Shf}(q,l)} \mathbb{I}[s(q, c) > s(\tilde{q}, c)] \tag{30}$$