A NEURAL SUBGRAPH MATCHING MODEL FOR GRAPH SEQUENCE RETRIEVAL

Anonymous authors
Paper under double-blind review

ABSTRACT

Subgraph-isomorphism-based retrieval traditionally compares a query graph against individual corpus graphs, whereas many relational systems are observed as evolving graph sequences. In this work, we focus on graph-sequence retrieval, where relevance is defined by the presence of an order-preserving subsequence in the corpus sequence, such that each graph in the subsequence contains the corresponding query graph as a subgraph. A brute-force approach would require solving subgraph-matching problems across query–corpus timestep pairs and then searching over order-preserving subsequences, leading to a costly combinatorial procedure. A straightforward neural adaptation would apply an independent graph-matching model to all timestep pairs before using a sequence-level matcher, but this ignores the temporally correlated structure of evolving graph sequences. We formalize this problem as Subgraph Isomorphism-based Graph Subsequence Matching (SIGMA), where relevance depends jointly on a shared node alignment and an order-preserving timestep alignment. We propose SIGMANET, an end-to-end neural retrieval model that replaces discrete optimization with differentiable node and timestep alignments. Across five graph-sequence retrieval benchmarks, SIGMANET consistently outperforms dynamic graph models and graph-matching baselines adapted to graph-sequence retrieval using sequence matchers.

1 INTRODUCTION

In traditional subgraph-matching–based graph retrieval, we are given a large corpus of graphs and aim to retrieve those that are relevant, *i.e.*, graphs that contain a given query graph as a subgraph. In contrast, graph sequence retrieval operates over a corpus of graph sequences $S_c = (G_{c,1}, G_{c,2}, \ldots)$, rather than individual graphs. A corpus sequence S_c is relevant to a query sequence $S_q = (G_{q,1}, G_{q,2}, \ldots)$ if the query embeds as a subsequence under subgraph isomorphism. Formally, S_c is relevant to S_q if there exist indices $\mu(1) < \mu(2) < \cdots$ such that each query graph $G_{q,t}$ is subgraph-isomorphic to the corresponding corpus graph $G_{c,\mu(t)}$. We call this problem **S**ubgraph **I**somorphism based **G**raph subsequence **MA**tching (SIGMA). Similar to the subgraph matching problem, SIGMA is NP-hard too.

Graph sequence retrieval has potential applications in several domains. In computational chemistry, a reaction mechanism can be represented as a sequence of molecular graphs (Unsleber, 2023; Plehiers et al., 2018; Phan et al., 2025); determining whether a query motif of functional groups appears consistently across successive intermediates is naturally a subsequence-matching problem over graphs. In video understanding (Liu et al., 2022; Zhang et al., 2020; Carreira et al., 2017), each frame can be represented as a scene graph (Ji et al., 2020; Li et al., 2024a), and the goal is to retrieve a sequence of scene graphs corresponding to an action sequence. Driven by this motivation, we develop neural models for the subgraph-matching-based graph sequence retrieval task.

Neural models for traditional subgraph matching based graph retrieval (Li et al., 2019; Roy et al., 2022; Ramachandran et al., 2024; Ranjan et al., 2022) typically aim to learn node-to-node correspondences, using differentiable alignment networks, from the distant supervision of the relevance labels. The problem of graph sequence retrieval involves two alignment variables: **(1)** node alignment, which captures node-to-node correspondences between two candidate graphs from query and corpus sequences; and, **(2)** sequence alignment, which captures the mapping between time steps across query and corpus, and such mapping must form a strictly increasing sequence. A key challenge is that these choices are mutually dependent: node alignment quantifies the amount of subgraph matching

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

between query-corpus pairs, which is then used for sequence alignment. On the other hand, the sequence alignment determines which graph pairs must be matched under a common node alignment.

1.1 Our contributions

Responding to the above challenges, we present SigmaNet, a neural retrieval model for SIGMA based graph sequence retrieval that learns a shared node alignment and an order-preserving sequence alignment using a neural network. Specifically, we make the following contributions.

Introducing a novel graph sequence retrieval problem We introduce graph sequence retrieval, a novel retrieval problem based on SIGMA. Unlike traditional subgraph matching-based graph retrieval, which compares a single query graph against a single corpus graph, SIGMA asks whether an ordered query graph sequence appears embedded *within* a corpus graph sequence under subgraph isomorphism at every matched timestep. This formulation couples node-level and sequence-level alignment, making relevance depend on both jointly.

Combinatorial formulation of SIGMA We start from the quadratic assignment problem (QAP) for subgraph isomorphism on a single graph pair and lift it to graph sequences. Driven by several practical applications (Ji et al., 2020; Plehiers et al., 2018; Phan et al., 2025), we posit that the node correspondences are preserved within the same sequence. This allows us to cast SIGMA as a QAP over the node-to-node permutation matrix and the sequence alignment map, subject to the constraint that the sequence alignment map induces a monotone injective function. This monotone injectivity ensures an order-preserving match between the query and the corpus graph sequence.

Differentiable network for SIGMA We propose SigmaNet, a neural network that replaces the combinatorial optimization with a differentiable layer, which is end-to-end trainable from distant supervision of binary relevance labels. It operates in three components– **(1)** We first develop a sequence-aware graph encoder that captures both structural and temporal context in node representations. **(2)** Instead of jointly optimizing the node alignment matrix and the staircase matrix, we infer the node alignment by applying a Sinkhorn network on the sequence-aware graph embeddings computed as above. **(3)** We obtain a differentiable relaxation of the staircase matrix by factoring it into a row-selector mask and a right-monotone permutation matrix (Kohli et al., 2026). This provides a differentiable approximation of the staircase matrix through Sinkhorn normalization of the underlying permutation matrix.

We combine the sequence-aware embeddings, the approximate node alignment matrix, and the approximate sequence alignment matrix to relax the combinatorial sequence distance into a neural sequence distance. Our experiments on five different datasets show that SigmaNet performs better than several baselines.

2 Preliminaries

Notations Let $[n] = \{1, \dots, n\}$ for $n \in \mathbb{N}$. We denote a query graph sequence by an ordered collection of graphs $S_q = (G_{q,1}, \dots, G_{q,\ell_q})$ and a corpus graph sequence by $S_c = (G_{c,1}, \dots, G_{c,\ell_c})$, with ℓ_q and ℓ_c being their corresponding sequence lengths. Here, $G_{q,t} = (V_{q,t}, E_{q,t})$ and $G_{c,\tau} = (V_{c,\tau}, E_{c,\tau})$ denote the query and corpus graphs at positions $t \in [\ell_q]$ and $\tau \in [\ell_c]$, respectively. In our work, we assume node identities are persistent, *i.e.*, $V_{q,t} = V_q$ and $V_{c,\tau} = V_c$. We additionally pad the graphs to $n = \max(|V_q|, |V_c|)$ nodes, so that every adjacency matrix has the same dimension for $G_{q,t}$ and $G_{c,\tau}$, and the adjacency matrices satisfy $\boldsymbol{A}_{q,t}, \boldsymbol{A}_{c,\tau} \in \{0,1\}^{n \times n}$.

We write Π_n for the set of $n \times n$ permutation matrices and $\mathcal{B}_n$ for the Birkhoff polytope of $n \times n$ doubly-stochastic matrices (or soft permutation matrices). We denote $\boldsymbol{P} \in \Pi_n$ or $\boldsymbol{P} \in \mathcal{B}_n$ to a permutation matrix or a doubly stochastic matrix. We write $[\,\cdot\,]_+ = \max\{\cdot, 0\}$ for the elementwise hinge (ReLU) function.

We write the collection of corpus sequences as $\{S_c\}$ indexed by $c \in C$ and the query sequences as $\{S_q\}$ indexed by $q \in Q$. We denote the relevance of corpus sequences S_c to query S_q as $y_{q,c}$: here, $y_{q,c} = +1$ if S_c is relevant to S_q and $y_{q,c} = -1$, otherwise. Given a query S_q, the sets $C_{\checkmark}$ and $C_{\times}$ collect the indices of all relevant and non-relevant corpus sequences, respectively, i.e., $C_{\checkmark} = \{c : y_{q,c} = +1\}$ and $C_{\times} = C \setminus C_{\checkmark}$.

Traditional subgraph matching Traditional subgraph-matching-based graph retrieval (Roy et al., 2022; Ramachandran et al., 2024; Ranjan et al., 2022; Li et al., 2019) operates on individual graphs

rather than sequences: the corpus is a set $\{G_c\}$ of single graphs, and the query is a single graph G_q, in contrast to our setting where both query and corpus are sequences of graphs of the form $S_c = (G_{c,\tau} : \tau \in [\ell_c])$ for corpus and $S_q = (G_{q,t} : t \in [\ell_q])$. The task reduces to retrieving the top-K corpus graphs relevant to G_q and the relevance is defined pairwise between *graphs (rather than sequences)* as $y = +1$ if $G_q \subseteq G_c$ (i.e., G_q is isomorphic to a subgraph of G_c) and $y = -1$ otherwise.

The condition $G_q \subseteq G_c$ holds iff there exists a node permutation $\boldsymbol{P} \in \Pi_n$ such that $\boldsymbol{A}_q \leq \boldsymbol{P}\boldsymbol{A}_c\boldsymbol{P}^\top$ holds elementwise. This induces the asymmetric relevance distance $\Delta(G_q, G_c) = \min_{P\in\Pi_n} \|[\boldsymbol{A}_q - \boldsymbol{P}\boldsymbol{A}_c\boldsymbol{P}^\top]_+\|_{1,1}$ which is zero if and only if the subgraph isomorphism condition holds.

Staircase matrix A binary matrix $\boldsymbol{B}$ is called a staircase matrix if (1) $\boldsymbol{B}$ has exactly one 1 per row; and (2) the column index of that 1 is strictly increasing down the rows. Equivalently, $\boldsymbol{B}$ encodes an order-preserving injection $\mu : [\ell_q] \to [\ell_c]$ via $B[t, \tau] = \mathbb{I}[\tau = \mu(t)]$.

3 Subgraph Isomorphism-based Graph Subsequence Matching

In this section, we first introduce the new problem of subgraph isomorphism-based graph subsequence matching (SIGMA) and then formulate a combinatorial optimization framework for SIGMA.

3.1 Introducing SIGMA

We first provide a formal definition of SIGMA as follows.

Definition 1 (SIGMA ($\subseteq_{[b_1,b_2]}$))**.** Given two graph sequences $S_q = (G_{q,t} = (V_q, E_{q,t}) : t \in [\ell_q])$ and $S_c = (G_{c,\tau} = (V_c, E_{c,\tau}) : \tau \in [\ell_c])$ and bounds $1 \leq b_1 \leq b_2 \leq \infty$, where the node identities are persistent across time (as $V_{\bullet,t} = V_\bullet$). We say S_q admits a *subgraph isomorphism based graph sequence match* (SIGMA) with S_c, written as $S_q \subseteq_{[b_1,b_2]} S_c$, if there exist indices $1 \leq \tau_1, \cdots, \tau_{\ell_q} \leq \ell_c$ such that $G_{q,t} \subseteq G_{c,\tau_t}$, for every $t \in [\ell_q]$, and $b_1 \leq \tau_{t+1} - \tau_t \leq b_2$ for every $t \in [\ell_q - 1]$ *with a common node alignment between V_q and V_c across the matches.*

In words, S_q admits a SIGMA with S_c or $S_q \subseteq_{[b_1,b_2]} S_c$ if the graphs in S_q are subgraphs of the graphs in S_c present as an ordered subsequence with the matched graph pairs sharing a *common node alignment*. Successive gaps must be bounded above by b_2 and below by b_1.

The shared alignment setting is motivated by real-world applications (Joung et al., 2025; Plehiers et al., 2018; Phan et al., 2025; Fu et al., 2022; Ji et al., 2020) in which graph sequences capture the temporal evolution of objects and relations while node correspondences remain invariant over time. For example, the corpus may consist of chemical-reaction sequences, while the query represents the evolution of a particular substructure. Here, graph-sequence pairs that admit SIGMA yield a single node permutation $\boldsymbol{P}$ that consistently aligns corresponding nodes across time. However, our work can be used to generalize SIGMA where this alignment varies across time.

SIGMA based graph sequence retrieval We are given a collection of graph sequences $\{S_c\}$, indexed by C. Given a query graph sequence S_q, we assign binary relevance labels $y_{q,c} = +1$ if $S_q \subseteq_{[b_1,b_2]} S_c$, and $y_{q,c} = -1$ otherwise. Our goal is to retrieve from the corpus $\{S_c\}$ sequences relevant to S_q. We do this by deriving a distance function for SIGMA and learning a differentiable surrogate for it.

3.2 Combinatorial optimization of SIGMA

Quadratic assignment problem We develop the quadratic assignment problem of SIGMA step by step, starting with the simplest case when $\ell_q = \ell_c > 1$ with $b_1 = b_2 = 1$. Here, the task reduces to subgraph isomorphism-based graph *sequence* matching rather than *subsequence* matching, *i.e.*, every query graph must be a subgraph of the corpus graph at the *same* timestep, with no freedom in choosing which corpus timestep to align to. The task of determining whether $S_q \subseteq_{[b_1,b_2]} S_c$ then reduces to checking whether $G_{q,t} \subseteq G_{c,t}$ holds for every $t \in [\ell_q] = [\ell_c]$ under a shared alignment. Aggregating the QAP condition on subgraph isomorphism for individual graphs across all timesteps, the distance between the two graph sequences takes the form

$$\Delta(S_q, S_c) = \min_{\boldsymbol{P}\in\Pi_n} \textstyle\sum_{t\in[\ell_q]} \|[\boldsymbol{A}_{q,t} - \boldsymbol{P}\boldsymbol{A}_{c,t}\boldsymbol{P}^\top]_+\|_{1,1} \tag{1}$$

which vanishes precisely when each query graph admits a subgraph isomorphism into its corresponding corpus graph under the shared alignment $\boldsymbol{P}$.

Now suppose $\ell_q < \ell_c$, which reduces to the general setting of SIGMA. The key new challenge is the *order constraint* induced by $b_1 \geq 1$, *i.e.*, if $G_{q,1}$ is matched to some corpus graph G_{c,τ_1}, a valid match for $G_{q,2}$ must occur at a strictly later timestep $\tau_2 > \tau_1$, and the same applies to every subsequent query graph. The query sequence must therefore align with an *order-preserving subsequence* of the corpus sequence. We also enforce a notion of temporal compactness, that is, we require $\tau_2 - \tau_1 \leq b_2$ and similarly for subsequent timesteps.

In the $\ell_q = \ell_c$ case above, the matched subsequence was the entire corpus sequence — fixed and known. When $\ell_q < \ell_c$, the matched corpus subsequence is unknown. Choosing τ_1 constrains all later choices, since each subsequent index must be strictly larger than the previous one. The problem therefore becomes a joint search over two coupled variables, *i.e.*, the shared node alignment $\boldsymbol{P}$ and the strictly increasing timestep assignment $\tau_1 < \tau_2 < \cdots < \tau_{\ell_q}$.

To represent the strictly increasing timestep assignment, we introduce an order-preserving selection function $\mu : [\ell_q] \to [\ell_c]$, where $\mu(t)$ specifies the corpus timestep to which the t-th query graph is compared. Define $\delta_t = \mu(t+1) - \mu(t)$ as the matching gap between successive matches. Then the *relevance distance* between the query and corpus sequences is the aggregated distance between the graphs in S_q and the graphs in the corresponding corpus subsequence in S_c:

$$\Delta(S_q, S_c) = \min_{\boldsymbol{P}\in\Pi_n,\mu} \sum_{t\in[\ell_q]} \left\| \left[\boldsymbol{A}_{q,t} - \boldsymbol{P}\boldsymbol{A}_{c,\mu(t)}\boldsymbol{P}^\top\right]_+ \right\|_{1,1} \text{ so that } b_1 \leq \delta_t \leq b_2, t \in [\ell_q - 1]. \quad (2)$$

The query is an ordered subsequence of the corpus sequence with a common node alignment $\boldsymbol{P}$ precisely when $\Delta(S_q, S_c) = 0$. In general, $\Delta(S_q, S_c)$ defines an asymmetric relevance distance measuring how close the query is to being embeddable in the corpus subject to temporal compactness constraints. The $\ell_q = \ell_c$ case forces $\mu(t) = t$ and recovers the optimization problem (1); further setting $\ell_q = \ell_c = 1$ collapses the sum to the single-graph distance $\Delta(G_q, G_c)$ defined in Section 2.

Representing order-preserving injection using a staircase matrix The optimization problem (2) involves both a permutation matrix and an order-preserving injective function. Such a form may not be amenable for designing a neural surrogate, which is our primary objective. We therefore reformulate (2) so that all unknowns are matrix-valued. To this end, we replace the order-preserving selection function μ with a *staircase matrix* $\boldsymbol{B}$.

Let $\boldsymbol{B} \in \{0,1\}^{\ell_q \times \ell_c}$ be a matrix with $\boldsymbol{B}[t, \mu(t)] = 1$ and zeros elsewhere. By construction, $\boldsymbol{B}$ has exactly one 1 per row, and the column index of that 1 is strictly increasing down the rows, making $\boldsymbol{B}$ a staircase matrix (Section 2). Pre-multiplying any corpus-side vector $\boldsymbol{a} \in \mathbb{R}^{\ell_c}$ by $\boldsymbol{B}$ extracts precisely the entries selected by μ:

$$(\boldsymbol{B}\boldsymbol{a})[t] = \textstyle\sum_\tau \boldsymbol{B}[t,\tau]\,\boldsymbol{a}[\tau] = \boldsymbol{B}[t,\mu(t)]\,\boldsymbol{a}[\mu(t)] = \boldsymbol{a}[\mu(t)]. \quad (3)$$

More generally, left-multiplication by $\boldsymbol{B}$ restricts any quantity indexed over corpus timesteps $[\ell_c]$ to the subsequence induced by μ. The structure of $\boldsymbol{B}$ encodes order preservation directly. Using the property of $\boldsymbol{B}$ in Eq. (3), we express $\boldsymbol{A}_{c,\mu(t)}$ in Eq. (2) as the linear combination: $\boldsymbol{A}_{c,\mu(t)} = \sum_{\tau=1}^{\ell_c} \boldsymbol{B}[t,\tau]\boldsymbol{A}_{c,\tau}$. For notational ease, we also introduce a first-order difference operator $\boldsymbol{L}$ satisfying $(\boldsymbol{L}\boldsymbol{x})[t] = \boldsymbol{x}[t+1] - \boldsymbol{x}[t]$ and a vector $\boldsymbol{a}$ such that $\boldsymbol{a}[t] = t$. From Eq. (3), we have:

$$(\boldsymbol{L}\boldsymbol{B}\boldsymbol{a})[t] = \boldsymbol{a}[\mu(t+1)] - \boldsymbol{a}[\mu(t)] = \mu(t+1) - \mu(t) = \delta_t \quad (4)$$

Therefore, the staircase constraint in Eq. (2) holds if and only if every entry of $\boldsymbol{L}\boldsymbol{B}\boldsymbol{a}$ is strictly positive. This reduces the relevance distance computation in Eq. (2) as follows:

$$\Delta(S_q, S_c) = \min_{\boldsymbol{P}\in\Pi_n,\boldsymbol{B}} \sum_{t\in[\ell_q]} \left\| \left[\boldsymbol{A}_{q,t} - \boldsymbol{P} \sum_{\tau\in[\ell_c]} \boldsymbol{B}[t,\tau]\boldsymbol{A}_{c,\tau}\boldsymbol{P}^\top\right]_+ \right\|_{1,1} \text{ s.t. } b_1\mathbf{1} \leq \boldsymbol{L}\boldsymbol{B}\boldsymbol{a} \leq b_2\mathbf{1} \quad (5)$$

Since $\boldsymbol{B}$ has exactly one unit entry per row, the relevance distance admits a form where $\boldsymbol{B}$ can be factored out (see Appendix D).

$$\Delta(S_q, S_c) = \min_{\boldsymbol{P}\in\Pi_n,\boldsymbol{B}} \sum_{t\in[\ell_q]} \sum_{\tau\in[\ell_c]} \boldsymbol{B}[t,\tau] \left\| \left[\boldsymbol{A}_{q,t} - \boldsymbol{P}\boldsymbol{A}_{c,\tau}\boldsymbol{P}^\top\right]_+ \right\|_{1,1} \text{ s.t. } b_1\mathbf{1} \leq \boldsymbol{L}\boldsymbol{B}\boldsymbol{a} \leq b_2\mathbf{1} \quad (6)$$

4 SIGMANET MODEL

In this section, we describe SIGMANET, a neural network that approximates the combinatorial relevance distance in Eq. (6) using a differentiable surrogate $\widehat{\Delta}_\theta(S_q, S_c) \approx \Delta(S_q, S_c)$. Before presenting the model, we first discuss the key technical challenges.

Technical challenges The goal of SIGMANET is to replace the discrete optimization in Eq. (2) with a neural model trained from query–corpus relevance labels via a ranking objective, without explicit combinatorial search for training or inference.

(1) Sharing $\boldsymbol{P}$ across varying graph pairs. In the traditional subgraph isomorphism setting, a Sinkhorn layer takes GNN embeddings from the two graphs and aligns them to produce a doubly stochastic matrix (or soft permutation matrix) tailored to that specific pair. Here, the graphs $(G_{q,t}, G_{c,\tau})$ change across timesteps t and τ, but the ground-truth matched pairs (i.e. those with $B[t,\tau]=1$) have a common alignment $\boldsymbol{P}$. Hence, the network should be incentivised to produce an alignment jointly compatible with multiple graph pairs.

(2) Joint optimization over $\boldsymbol{P}$ and $\boldsymbol{B}$. The optimization problem (6) couples $\boldsymbol{P}$ and $\boldsymbol{B}$. A natural strategy is to alternate updates between their respective neural approximators, but this introduces a significant computational bottleneck, as each alternation requires a fresh forward pass, backward pass, and update step.

(3) Relaxing the staircase matrix. The OPAS construction (Kohli et al., 2026) expresses an order-preserving staircase alignment as a row-selected permutation matrix, enabling a differentiable relaxation through Sinkhorn normalization. However, OPAS considers sequence distances whose alignment costs are given directly between sequence elements. In SIGMA, the timestep-alignment costs are themselves induced by a shared node alignment $\boldsymbol{P}$ over the aligned pairs through a subgraph-matching objective. Thus the staircase relaxation must be coupled to graph-level asymmetric matching costs, rather than applied as a post-hoc sequence matcher.

We address these challenges through three corresponding components: **(1)** A sequence-aware graph encoder that produces node embeddings sensitive to the timestamp within the sequence. **(2)** A decoupled architecture that separates the computation of $\boldsymbol{P}$ and $\boldsymbol{B}$, without any alternating updates, and **(3)** A differentiable relaxation of the staircase matrix $\boldsymbol{B}$.

4.1 SEQUENCE-AWARE GRAPH ENCODER (GSEQENC)

Overview Here, we address technical challenge (1), *i.e.*, learning a single alignment matrix $\boldsymbol{P}$ that simultaneously aligns every graph in S_q with its counterpart in S_c. Since $\boldsymbol{P}$ must serve all timesteps at once, the input used to compute $\boldsymbol{P}$ must carry structural signals from every graph in both sequences. To this end, we design a neural sequence-aware graph encoder GSEQENC_θ that takes a graph sequence as input and produces node representations that capture both temporal evolution and local graph structure.

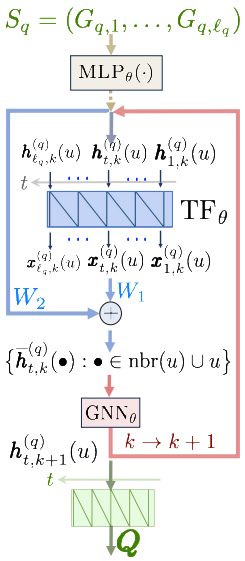

Figure 1: GSEQENC (S_q)

Given a query sequence $S_q = (G_{q,1}, \ldots, G_{q,\ell_q})$, $\text{GSEQENC}_\theta(S_q)$ computes d-dimensional node embeddings and aggregates them into a matrix $\boldsymbol{Q} = (Q_1, \ldots, Q_{\ell_q})$, where each $Q_t \in \mathbb{R}^{n\times d}$ contains the node embeddings of $G_{q,t}$. Similarly, we obtain $\boldsymbol{C} = (C_1, \ldots, C_{\ell_c})$ for S_c. GSEQENC consists of three components. **(a)** a causal transformer TF_θ **(b)** a graph neural network GNN_θ **(c)** a post-processing causal transformer TF'_θ. The causal transformer TF_θ performs *temporal contextualization* by mixing representations of the same node across timesteps, whereas the GNN performs *structural contextualization* by message passing through the edges. TF_θ and GNN_θ are tightly coupled: they alternately update node embeddings across temporal steps and message-passing layers, respectively. After this interleaved processing, TF'_θ computes the final embeddings.

Initialization We provide our analysis for query graph $G_{q,t}$ and likewise, we can extend for corpus $G_{c,\tau}$. We initialize the node embeddings $\boldsymbol{h}^{(q)}_{t,k}$ at $k=0$ of each graph in each query sequence individually using a simple multi-layer perceptron MLP as $\boldsymbol{h}^{(q)}_{t,k=0} = \text{MLP}_\theta(\text{feature}_u)$.

Temporal contextualization through causal transformer TF_θ We interleave each GNN layer with a causal decoder transformer TF_θ to perform temporal contextualization. For a query sequence S_q, let $\boldsymbol{h}^{(q)}_{t,k}(u) \in \mathbb{R}^{\dim_h}$ be the embedding of node u in graph $G_{q,t}$ after k GNN propagation layers.

Since node identities remain consistent across timesteps, we can form an *ordered stream* for each node $u \in V_q$ at layer k as $(\boldsymbol{h}^{(q)}_{1,k}(u), \ldots, \boldsymbol{h}^{(q)}_{\ell_q,k}(u))$ which traces node u's representations across the sequence. When dealing with graph sequences with node additions and deletions, we mask the above sequence to only contain the embeddings corresponding to the timesteps when node u is present.

We pass each stream through the causal transformer TF_θ, which restricts attention at timestep t to positions $1, \ldots, t$. The output of TF_θ at position t encodes how the node's local structure has evolved up to t, capturing the underlying temporal order. We compute these intermediate embeddings $\boldsymbol{x}^{(q)}_{t,k}(u)$ for different t as follows:

$$\boldsymbol{x}^{(q)}_{1,k}(u), \ldots, \boldsymbol{x}^{(q)}_{\ell_q,k}(u) = \mathrm{TF}_\theta\Big(\boldsymbol{h}^{(q)}_{1,k}(u), \ldots, \boldsymbol{h}^{(q)}_{\ell_q,k}(u)\Big) \quad \text{for all } u \in V_q \tag{7}$$

Skip connections Consider a relevant pair $S_q \subseteq_{[b_1,b_2]} S_c$ with $G_{q,t} \subseteq G_{c,\tau}$ for $\tau = \mu(t)$. The node alignment $\boldsymbol{P}$ obtained by solving the QAP on the single pair $(G_{q,t}, G_{c,\tau})$ may be the same as the optimal alignment for the full sequence-level QAP , Eq. (2). When this is the case, the model can benefit by focusing on the individual graph pair. In the embedding space for the query sequence S_q, this amounts to bypassing the transformer output $\boldsymbol{x}^{(q)}_{t,k}$ in Eq. (7) and feeding the GNN embeddings $\boldsymbol{h}^{(q)}_{t,k}$ directly into the next propagation layer. To allow the network to do that, we combine the transformer output $\boldsymbol{x}^{(q)}_{t,k}$ with the preceding GNN representation $\boldsymbol{h}^{(q)}_{t,k}$ via a learnable skip connection and obtain $\overline{\boldsymbol{h}}^{(q)}_{t,k}$.

$$\overline{\boldsymbol{h}}^{(q)}_{t,k}(u) = \boldsymbol{W}_1\, \boldsymbol{x}^{(q)}_{t,k}(u) + \boldsymbol{W}_2\, \boldsymbol{h}^{(q)}_{t,k}(u) \quad \text{for all } u \in V_q \tag{8}$$

where $\boldsymbol{W}_1, \boldsymbol{W}_2$ are learnable matrices. When $\boldsymbol{W}_1 \approx 0$, the layer reduces to a standard GNN step that ignores temporal context; when $\boldsymbol{W}_2 \approx 0$, it relies entirely on the transformer output.

Structural contextualization through message passing GNN GNN computes structurally contextual node embeddings using the temporally contextualized representations that already encode the node's sequence history. We update the embeddings $\boldsymbol{h}^{(q)}_{t,k}$ via a GNN layer that propagates information along the edges of each graph $G_{q,t}$ independently, as follows:

$$\boldsymbol{h}^{(q)}_{t,k+1}(u) = \mathrm{GNN}_\theta\Big(\overline{\boldsymbol{h}}^{(q)}_{t,k}(u),\ \Big\{\overline{\boldsymbol{h}}^{(q)}_{t,k}(v) : v \in \mathcal{N}(u, G_{q,t})\Big\}\Big) \quad \text{for all } u \in V_q \tag{9}$$

This produces updated node embeddings $\boldsymbol{h}^{(q)}_{t,k+1}$, which are then fed back into the transformer at layer $k{+}1$. That is, the GNN computes structurally contextual node embeddings by propagating over temporally contextualized representations that already encode each node's sequence history. The full encoder thus alternates K rounds of temporal contextualization (TF_θ), adaptive skip, and structural contextualization (GNN_θ).

Post processing transformer TF'_θ After K message-passing layers, the final GNN outputs are passed through one additional causal transformer TF'_θ, yielding the final node embeddings for S_q:

$$\boldsymbol{Q} := (\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_{\ell_q}) = \mathrm{TF}'_\theta(\boldsymbol{h}^{(q)}_{1,K}, \ldots, \boldsymbol{h}^{(q)}_{\ell_q,K}) \qquad \text{where} \quad \boldsymbol{Q}_t \in \mathbb{R}^{n \times d} \tag{10}$$

Similarly, we collect the final node embeddings for S_c in $\boldsymbol{C} := (\boldsymbol{C}_1, \ldots, \boldsymbol{C}_{\ell_c})$.

4.2 Staged decoupling of node alignment $\boldsymbol{P}$ and order preserving staircase matrix $\boldsymbol{B}$

Instead of directly learning this joint optimization, we use a staged approximation. We first estimate a sequence-level node alignment matrix $\boldsymbol{P}$, and then, in Section 4.3, we use $\boldsymbol{P}$ to solve for the sequence-alignment staircase matrix $\boldsymbol{B}$. This addresses challenge (2).

Since TF'_θ is causal, a node's last active state in the sequence summarises its temporal evolution. We therefore construct sequence-level node embeddings $\boldsymbol{Q}_{\text{last}}$ and $\boldsymbol{C}_{\text{last}}$ by taking the final active timestep embeddings for each node in query and corpus sequence, respectively. We project these representations using a Linear–ReLU–Linear (LRL_θ) and use their pairwise similarity matrix as input to a Sinkhorn layer:

$$\mathbf{Z} = \mathrm{LRL}_\theta(\boldsymbol{Q}_{\text{last}}) \cdot \mathrm{LRL}_\theta(\boldsymbol{C}_{\text{last}})^\top, \qquad \boldsymbol{P} = \mathrm{Sinkhorn}(\mathbf{Z}).$$

The resulting $\boldsymbol{P} \in \mathcal{B}_n$ is a shared soft node alignment reused across all query–corpus timestep pairs.

4.3 Differentiable approximation of staircase matrix $\boldsymbol{B}$

Order-preserving penalty Having addressed the first two challenges, we now turn to Challenge 3: learning $\boldsymbol{B}$. Building on Kohli et al. (2026), we exploit the fact that any valid staircase matrix admits the factorization $\boldsymbol{B} = \boldsymbol{M}\,\boldsymbol{R}$, where $\boldsymbol{R} \in \Pi_{\ell_c}$ is a $\ell_c \times \ell_c$ permutation matrix and $\boldsymbol{M} \in \{0,1\}^{\ell_q \times \ell_c}$ is a fixed row-selector formed from the first ℓ_q rows of the identity matrix. This decomposition reduces the problem to learning $\boldsymbol{R}$, to which we can directly apply Sinkhorn iterations to obtain a differentiable relaxation that yields a surrogate for the staircase matrix.

However, the reverse is not true. For every permutation matrix $\boldsymbol{R}$, the product $\boldsymbol{M} \cdot \boldsymbol{R}$ may not be a valid staircase matrix. In fact, a reverse staircase, where the column index of the 1 decreases down the rows, admits the same factorization. Hence we need an additional constraint for monotonicity, enforced by $b_1\mathbf{1} \le \boldsymbol{LBa} \le b_2\mathbf{1}$ from Eq. (2). Finite b_2 additionally enforces temporal compactness.

Neural cost matrix Given a fixed soft node alignment $\boldsymbol{P}$, we relax the combinatorial objective in Eq. (6) into a linear assignment problem between sequence of node embeddings $\boldsymbol{Q}_t$ and $\boldsymbol{C}_\tau$. Hence, the underlying relaxed optimization problem becomes

$$\min_{\boldsymbol{R}} \sum_{t,\tau} (\boldsymbol{MR})[t,\tau] \|[\boldsymbol{Q}_t - \boldsymbol{P}\boldsymbol{C}_\tau]_+\|_{1,1}, \qquad \text{subject to,} \quad b_1\mathbf{1} \le \boldsymbol{LBa} \le b_2\mathbf{1} \tag{11}$$

If we define a matrix $\mathrm{cost}(\boldsymbol{P})$ with $\mathrm{cost}(\boldsymbol{P})[t,\tau] = \left\|\left[\boldsymbol{Q}_t - \boldsymbol{P}\,\boldsymbol{C}_\tau\right]_+\right\|_{1,1}$, then by using Lagrangian multipliers $\boldsymbol{\lambda_1}, \boldsymbol{\lambda_2}$, we obtain:

$$\min_{\boldsymbol{R} \in \Pi_{\ell_c}} \langle \boldsymbol{MR},\, \mathrm{cost}(\boldsymbol{P}) \rangle_F + \langle \boldsymbol{\lambda}_1, b_1\mathbf{1} - \boldsymbol{LMRa} \rangle + \langle \boldsymbol{\lambda}_2, \boldsymbol{LMRa} - b_2\mathbf{1} \rangle \tag{12}$$

The first term favors low subgraph-matching cost, while the second and third terms penalize successive differences that fall below b_1 or exceed b_2, respectively. Appendix D derives an equivalent linear maximization form over $\boldsymbol{R}$

$$\max_{\boldsymbol{R} \in \Pi_{\ell_c}} \langle \boldsymbol{R}, \boldsymbol{\Omega} \rangle, \qquad \text{where } \boldsymbol{\Omega} = \boldsymbol{M}^\top \big(\boldsymbol{L}^\top (\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_2)\, \boldsymbol{a}^\top - \mathrm{cost}(\boldsymbol{P}) \big) \in \mathbb{R}^{\ell_c \times \ell_c} \tag{13}$$

We apply Sinkhorn to $\boldsymbol{\Omega}$, obtaining the soft $\boldsymbol{R}$, and recover $\boldsymbol{B} = \boldsymbol{M} \cdot \boldsymbol{R}$. This completes the differentiable relaxation of the staircase constraint, resolving Challenge 3. The parameters $\boldsymbol{\lambda_1}$ and $\boldsymbol{\lambda_2}$ are computed from a timestep-level similarity adapter ψ, which attention-pools node embeddings $\boldsymbol{Q}_t$ and $\boldsymbol{C}_\tau$ and forms pairwise query–corpus similarities (see Appendix E)

$$\boldsymbol{\lambda}_i = \mathrm{LagNet}_\theta^{(i)}\big(\psi(\boldsymbol{Q}, \boldsymbol{C})\big) \in \mathbb{R}_{\ge 0}^{\ell_q - 1}, \quad \text{for } i \in \{1,2\} \tag{14}$$

We put Eq. (14) and the Sinkhorn network guided solution of Eq. (13) into the objective of Eq. (12) to obtain the relevance distance surrogate $\widehat{\Delta}_\theta(S_q, S_c)$. Given training data $\{(S_q, S_c, y_{q,c})\}$, we estimate θ using the pairwise ranking objective described in Appendix E.

Imposing constraints via a Lagrangian penalty ensures the objective is linear in $\boldsymbol{R}$, allowing us to cast the problem as a LAP and obtain a differentiable relaxation for $\boldsymbol{R}$. We could also use a hinge-based penalty instead, *i.e.*, one that avoids rewarding slack in the constraints. But that would prevent us from casting Eq. (13) using an LAP and subsequent Sinkhorn iterations. We instead use them for relevance score calculation as follows,

$$\mathrm{score}(S_q, S_c) = -\beta_1 \langle \boldsymbol{B}, \mathrm{cost}(\boldsymbol{P}) \rangle_F - \beta_2 \boldsymbol{\lambda}_1^\top [b_1\mathbf{1} - \boldsymbol{LBa}]_+ - \beta_3 \boldsymbol{\lambda}_2^\top [\boldsymbol{LBa} - b_2\mathbf{1}]_+ \tag{15}$$

Given training data $\{(S_q, S_c, y_{q,c})\}$, we estimate θ using the ranking loss (Appendix E).

5 Experiments

Datasets We use five graph sequence retrieval datasets, *viz.*, **(1)** PaRoutes (Genheden et al., 2022) **(2)** ICEWS-18 (ICEWS) (Jin et al., 2020; Li et al., 2021; Boschee et al., 2015), **(3)** DPPIN-Ito (Ito) (Fu et al., 2022) **(4)** DPPIN-Uetz (Uetz) (Fu et al., 2022). **(5)** FlowER (Joung et al., 2025) Details are in Appendix E.

Baselines The key technical feature of our work, *i.e.*, jointly computing node-to-node correspondences across graphs and aligning their associated sequences, is scarce in the literature. Hence, we adapt existing works from other domains for comparison. Specifically, we use eight baselines, *viz.*, **(1)** GEN (Li et al., 2019), **(2)** GREED (Ranjan et al., 2022), **(3)** SimGNN (Bai et al., 2019), **(4)** Neuromatch (Ying et al., 2020), **(5)** IsoNet (Roy et al., 2022), **(6)** GraphSSM (Li et al., 2024b), **(7)** MaskDGNN (He et al., 2025) and **(8)** UnderGS (Wu et al., 2026). The first five baselines are adapted from graph matching literature, while the last three are dynamic graph embedders. Baselines use the SDTW sequence matcher.

Baselines	MAP					MRR				
	PaRoutes	ICEWS	Ito	Uetz	FlowER	PaRoutes	ICEWS	Ito	Uetz	FlowER
GEN	0.757	0.478	0.753	0.841	0.699	<u>0.866</u>	0.693	0.976	0.955	0.872
GREED	0.700	0.497	0.742	0.850	0.606	0.772	<u>0.748</u>	0.943	0.943	0.747
SimGNN	0.725	0.473	0.661	0.800	0.615	0.826	0.671	0.928	0.925	0.777
Neuromatch	0.714	0.486	0.693	0.801	0.688	0.766	0.711	0.970	0.908	0.871
IsoNet	<u>0.761</u>	<u>0.513</u>	<u>0.838</u>	<u>0.936</u>	<u>0.793</u>	0.865	0.678	<u>0.987</u>	**1.000**	**0.915**
GraphSSM	0.694	0.461	0.735	0.862	0.761	0.800	0.675	0.923	0.939	<u>0.886</u>
MaskDGNN	0.586	0.456	0.732	0.829	0.648	0.692	0.721	0.956	0.889	0.709
UnderGS	0.573	0.437	0.700	0.782	0.667	0.745	0.726	0.922	0.870	0.783
SIGMANET	**0.818**	**0.527**	**0.851**	**0.948**	**0.812**	**0.938**	**0.783**	**0.991**	<u>0.979</u>	**0.915**

Table 2: Retrieval performance measured in terms of Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR) for SIGMANET and the baselines using *SDTW* for sequence matching. SIGMANET learns a shared node alignment and an ordered timestep alignment. SIGMANET achieves the highest MAP across all datasets and highest MRR on four datasets. **Bold** denotes the best and <u>underline</u> denotes the second-best value in each column.

Evaluation Given a query set Q and a fixed corpus C, we partition Q into training, validation, and test sets using a 60%/15%/25% split. For each query sequence S_q, we compute the distance $-\text{score}(S_q, S_c)$ to every corpus sequence $S_c \in C$ and rank the corpus sequences in ascending order of distance. We obtain average precision (AP) and reciprocal rank (RR) for each ranked list, and average these metrics across all queries to report mean average precision (MAP) and mean reciprocal rank (MRR), respectively.

5.1 RESULTS

Comparison with baselines We first compare SIGMANET against all baselines. Table 2 summarizes the results. We make the following observations. **(1)** SIGMANET achieves the best MAP across all datasets and the highest MRR on all but Uetz; **(2)** Among the baselines, IsoNet is consistently the strongest baseline across all the datasets. This is because its node-level asymmetric hinge cost and explicit node alignment planning module. GEN also uses asymmetric hinge cost on pooled embeddings and therefore shows good performance compared to other baselines. **(3)** SimGNN performs poorly because of its scoring function is not tailored to subgraph matching, while its graph-level embeddings and plain GCN layers may not adequately capture structural information. **(4)** GraphSSM achieves the highest MAP among the dynamic graph embedding-based baselines on every dataset. It updates graph embeddings while retaining information from previous snapshots. In contrast, UnderGS retains only selected historical neighbors, which may discard useful information about structural evolution.

Ablation on temporal contextualization TF_θ To assess the effect of temporal contextualization TF_θ, we compare SIGMANET with a variant that omits TF_θ. Table 3 reports performance in terms of MAP and MRR. We observe that removing TF_θ from SIGMANET lowers the MAP on four out of five datasets and MRR on all five. On Ito, MAP marginally improves without TF_θ. If node features remain unchanged across timesteps, their initial embeddings become identical. Applying TF_θ before the GNN update produces temporally informed node representations.

Mean Average Precision (MAP)					
	PaRoutes	ICEWS	Ito	Uetz	FlowER
SIGMANET (No TF_θ)	<u>0.807</u>	<u>0.489</u>	**0.853**	<u>0.928</u>	<u>0.792</u>
SIGMANET	**0.818**	**0.527**	<u>0.851</u>	**0.948**	**0.812**
Mean Reciprocal Rank (MRR)					
	PaRoutes	ICEWS	Ito	Uetz	FlowER
SIGMANET (No TF_θ)	<u>0.896</u>	<u>0.692</u>	<u>0.987</u>	<u>0.973</u>	<u>0.893</u>
SIGMANET	**0.938**	**0.783**	**0.991**	**0.979**	**0.915**

Table 3: Effect of temporal contextualization TF_θ. **Bold** (<u>underline</u>) denotes the (second) best.

Effect of position of temporal contextualizer TF_θ In SIGMANET, we place TF_θ before each GNN update step and use a skip connection. To understand the impact of these two components, we place TF_θ before (PRE) or after (POST) each GNN update, with (skip) or without a skip connection (No-skip). Table 4 summarizes the MAP and MRR results. We make the following observations. **(1)** PRE + skip has the highest MAP on three of five datasets and the highest MRR on four of five datasets. **(2)** With a skip connection, PRE outperforms POST on four of the five datasets. Applying TF_θ *before* message passing lets the GNN capture richer temporal information than applying it after message passing. **(3)** Adding the skip connection to PRE improves MAP on ICEWS, Uetz, and FlowER. This is because the skip connection preserves the current node representation alongside its temporal context.

Configuration	MAP					MRR				
	PaRoutes	ICEWS	Ito	Uetz	FlowER	PaRoutes	ICEWS	Ito	Uetz	FlowER
No-TF$_\theta$	0.807	0.489	0.853	0.928	0.792	0.896	0.692	0.987	0.973	0.893
POST + No-skip	0.767	0.491	0.831	0.937	0.784	0.840	0.640	0.987	0.966	0.868
PRE + No-skip	**0.819**	0.462	0.852	0.935	0.797	0.904	0.623	0.978	**0.993**	0.848
POST + skip	0.810	0.491	**0.857**	0.913	0.809	0.903	0.678	0.987	0.961	0.888
PRE + skip (*default*)	0.818	**0.527**	0.851	**0.948**	**0.812**	**0.938**	**0.783**	**0.991**	0.979	**0.915**

Table 4: MAP and MRR performance of SIGMANET for different positions of TF$_\theta$ and skip-connection settings. PRE and POST place TF$_\theta$ before and after each GNN update, respectively. PRE + skip has the highest average MAP and MRR, leading on three datasets in MAP and four datasets in MRR. **Bold** denotes the best and underline denotes the second-best value in each column.

Ablation on Staircase aligner To understand the impact of Staircase aligner (Section 4.3), we compare SIGMANET with and without the Staircase aligner (SA). Table 5 shows that removing the Staircase aligner from SIGMANET significantly lowers the MAP and MRR across all datasets.

Mean Average Precision (MAP)					
	PaRoutes	ICEWS	Ito	Uetz	FlowER
SIGMANET (No SA)	0.732	0.505	0.824	0.879	0.796
SIGMANET	**0.818**	**0.527**	**0.851**	**0.948**	**0.812**
Mean Reciprocal Rank (MRR)					
	PaRoutes	ICEWS	Ito	Uetz	FlowER
SIGMANET (No SA)	0.770	0.702	0.984	0.976	0.908
SIGMANET	**0.938**	**0.783**	**0.991**	**0.979**	**0.915**

Table 5: Effect of Staircase aligner (SA). **Bold** (underline) denotes the (second best).

Shared $\boldsymbol{P}$ versus distinct $\boldsymbol{P}$ We compare SIGMANET with an alternative that estimates a distinct node alignment $\boldsymbol{P}_{t,\tau}$ for each query–corpus graph pair. Table 6 summarizes the MAP results. We observe that SIGMANET with a shared $\boldsymbol{P}$ achieves higher MAP on all five datasets. A separate $\boldsymbol{P}_{t,\tau}$ increases expressivity, but the shared $\boldsymbol{P}$ enhances inductive bias, since our datasets are motivated by applications where node identity and alignment persist across timesteps. Moreover, a separate $\boldsymbol{P}_{t,\tau}$ results in worse retrieval performance with the additional cost of computing and storing $\ell_q \times \ell_c$ node alignment matrices.

Methods	PaRoutes	ICEWS	Ito	Uetz	FlowER
Distinct $\boldsymbol{P}$	0.814	0.503	0.848	0.943	0.779
Shared $\boldsymbol{P}$	**0.818**	**0.527**	**0.851**	**0.948**	**0.812**

Table 6: Shared $\boldsymbol{P}$ vs distinct $\boldsymbol{P}$. **Bold** denotes the best value in each column.

Refinement of the shared node alignment We examine how the learned alignment changes across propagation layers. For sequence pairs with a shared alignment $\boldsymbol{P}^\star$ found by a combinatorial solver, we compute the soft node alignment $\boldsymbol{P}^{(k)}$ after each propagation layer and measure its distance to $\boldsymbol{P}^\star$. Figure 7 shows the empirical distribution of these errors across query–corpus pairs. As k increases, the errors decrease, suggesting that temporal contextualization and GNN message passing progressively refine the shared node alignment.

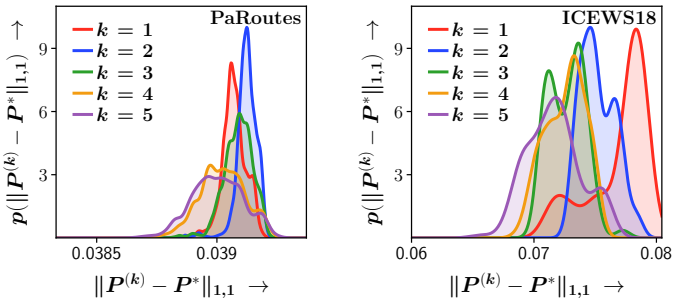

Figure 7: Empirical density of the node alignment error $\left\|\boldsymbol{P}^{(k)} - \boldsymbol{P}^\star\right\|_{1,1}$ across propagation layers k. Lower values indicate closer agreement with $\boldsymbol{P}^\star$.

6 CONCLUSION

We introduce a novel subgraph-isomorphism-based graph-sequence retrieval problem, where a corpus sequence is considered a match if it contains an order-preserving subsequence such that each query graph is contained in its corresponding corpus graph (SIGMA). We formulate this as a combinatorial optimization problem over a shared node permutation and a monotone sequence aligner, and propose SIGMANET, a differentiable neural model that jointly learns approximate node and timestep alignment. Experiments across five benchmarks establish strong retrieval performance of SIGMANET. Ablations support that the gains stem from sequence-aware structural contextualization and explicit modeling of staircase-structured sequence alignment.

AI USE STATEMENT

In this work, we used generative AI tools for assistance in implementing methods, with generative AI tools providing assistance for coding, including building data-processing pipelines for datasets. We have not used generative AI tools for generating synthetic datasets, development of theoretical models or conceptual frameworks, proposing or refining hypothesis, assistance in formulating and proving mathematical claims, designing or providing feedback on research methodology or experimentation, supporting qualitative or thematic data analysis or interpretation of results. Use of generative AI for assistance with translation or cleaning and reformatting dataset(s) is not applicable to this work. Additionally, we used generative AI tools for assistance in editing the paper to improve readability, grammatical correctness and clarity of notation. We have reviewed all AI-assisted work. LLM generated code was verified, tested and reviewed by two authors and any edits proposed by generative AI were proofread by two authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Our work does not involve any human subjects. Our experiments use publicly available datasets and do not contain any personal or sensitive information. Although any retrieval method could be potentially misused, the experiments we conducted do not expose any privacy concerns. Apart from this, we are not aware of any additional ethical concerns arising from this work.

REPRODUCIBILITY STATEMENT

We provide the necessary details for reproducibility of our experiments in this work. We describe the model architecture, training protocol and benchmark construction in the main paper and the appendix. We also provide the code and the datasets used in our experiments as supplementary material.

REFERENCES

Rdkit: Open-source cheminformatics. URL https://www.rdkit.org.

Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019.

Elizabeth Boschee, Jennifer Lautenschlager, Sean O'Brien, Steve Shellman, James Starz, and Michael Ward. ICEWS Coded Event Data, 2015.

Joao Carreira and Andrew Zisserman. Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset . In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems*, 2013.

Marco Cuturi and Mathieu Blondel. Soft-DTW: a differentiable loss function for time-series. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Dongqi Fu and Jingrui He. Dppin: A biological repository of dynamic protein-protein interaction network data. In *2022 IEEE International Conference on Big Data (Big Data)*, 2022.

Samuel Genheden and Esben Bjerrum. Paroutes: towards a framework for benchmarking retrosynthesis route predictions. *Digital Discovery*, 2022.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017.

Yiming He, Xiang Li, Zhongying Zhao, Haobing Liu, Peilan He, and Yanwei Yu. Maskdgnn: self-supervised dynamic graph neural networks with activeness-aware temporal masking. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, 2025.

Jingwei Ji, Ranjay Krishna, Li Fei-Fei, and Juan Carlos Niebles. Action genome: Actions as compositions of spatio-temporal scene graphs. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Woojeong Jin, Meng Qu, Xisen Jin, and Xiang Ren. Recurrent event network: Autoregressive structure inference over temporal knowledge graphs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

Joonyoung F. Joung, Mun Hong Fong, Nicholas Casetti, Jordan P. Liles, Ne S. Dassanayake, and Connor W. Coley. Electron flow matching for generative reaction mechanism prediction. *Nature*, 2025.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Bhavya Kohli, Soutrik Sarangi, Aziz Shameem, and Abir De. Learning right monotone permutation matrices for neural subsequence search. In *The 29th International Conference on Artificial Intelligence and Statistics*, 2026.

Hongsheng Li, Guangming Zhu, Liang Zhang, Youliang Jiang, Yixuan Dang, Haoran Hou, Peiyi Shen, Xia Zhao, Syed Afaq Ali Shah, and Mohammed Bennamoun. Scene graph generation: A comprehensive survey. *Neurocomputing*, 2024a.

Jintang Li, Ruofan Wu, Xinzhou Jin, Boqun Ma, Liang Chen, and Zibin Zheng. State space models on temporal graphs: A first-principles study. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024b.

Yujia Li, Richard Zemel, Marc Brockschmidt, and Daniel Tarlow. Gated graph sequence neural networks. In *Proceedings of ICLR'16*, 2016.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Zixuan Li, Xiaolong Jin, Wei Li, Saiping Guan, Jiafeng Guo, Huawei Shen, Yuanzhuo Wang, and Xueqi Cheng. Temporal knowledge graph reasoning based on evolutional representation learning. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021.

Yang Liu, Keze Wang, Lingbo Liu, Haoyuan Lan, and Liang Lin. Tcgl: Temporal contrastive graph for self-supervised video representation learning. *IEEE Transactions on Image Processing*, 2022.

Ciaran McCreesh, Patrick Prosser, and James Trimble. The glasgow subgraph solver: Using constraint programming to tackle hard subgraph isomorphism problem variants. In *Graph Transformation: 13th International Conference, ICGT*, 2020.

Gonzalo Mena, David Belanger, Scott Linderman, and Jasper Snoek. Learning latent permutations with gumbel-sinkhorn networks. In *International Conference on Learning Representations*, 2018.

Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 1970.

Tieu-Long Phan, Klaus Weinbauer, Marcos E. González Laffitte, Yingjie Pan, Daniel Merkle, Jakob L. Andersen, Rolf Fagerberg, Christoph Flamm, and Peter F. Stadler. Syntemp: Efficient extraction of graph-based reaction rules from large-scale reaction databases. *Journal of Chemical Information and Modeling*, 2025.

Pieter P. Plehiers, Guy B. Marin, Christian V. Stevens, and Kevin M. Van Geem. Automated reaction database and reaction network analysis: extraction of reaction templates using cheminformatics. *Journal of Cheminformatics*, 2018.

Ashwin Ramachandran, Vaibhav Raj, Indradyumna Roy, Soumen Chakrabarti, and Abir De. Iteratively refined early interaction alignment for subgraph matching based graph retrieval. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

Rishabh Ranjan, Siddharth Grover, Sourav Medya, Venkatesan Chakaravarthy, Yogish Sabharwal, and Sayan Ranu. GREED: A neural framework for learning graph distance functions. In *Advances in Neural Information Processing Systems*, 2022.

Indradyumna Roy, Venkata Sai Baba Reddy Velugoti, Soumen Chakrabarti, and Abir De. Interpretable neural subgraph matching for graph retrieval. In *Proceedings of the AAAI conference on artificial intelligence*, 2022.

H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1978.

Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020.

Richard Sinkhorn. Diagonal equivalence to matrices with prescribed row and column sums. *Amer. Math. Monthly*, 1967.

T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 1981.

Jan P. Unsleber. Accelerating reaction network explorations with automated reaction template extraction and application. *Journal of Chemical Information and Modeling*, 2023.

Danni Wu, Yuanyuan Xu, Xuemin Lin, Wenjie Zhang, and Ying Zhang. Understanding evolving graph structures for large discrete-time dynamic graph representation. *Proc. VLDB Endow.*, 2026.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

Rex Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. Neural subgraph matching. *CoRR*, abs/2007.03092, 2020.

Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: Graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022.

Jingran Zhang, Fumin Shen, Xing Xu, and Heng Tao Shen. Temporal reasoning graph for activity recognition. *IEEE Transactions on Image Processing*, 2020.

APPENDIX

A LIMITATIONS

In this work, we formulate the graph sequence matching problem (SIGMA) and build a corresponding neural retrieval model (SIGMANET) by relaxing the combinatorial optimization associated with SIGMA. We apply SIGMANET to datasets derived from real-world graph sequences, outperforming graph matching and dynamic graph variants adapted to SIGMA. However, our method has a few limitations:

(1) SIGMANET obtains the top-k relevant items by scoring each corpus element. As a result, it scales linearly with corpus size. An interesting avenue of future work could be to explore hashable score functions so that we may achieve sublinear top-k retrieval.

(2) SIGMANET corresponds to a late-interaction model as the query and corpus embeddings do not depend on each other and can be computed independently. An interesting avenue of future work may be to build an early interaction model for SIGMA, where query and corpus embeddings inform each other.

B BROADER IMPACT

Graph sequence retrieval can help search reaction databases for chemical transformations, protein-protein interaction databases for common interaction patterns, and video understanding to search for particular activities. Any system for information retrieval has the capability to be misused when applied to sensitive data. For instance, if graph sequences contain social interaction data, better retrieval systems can make it easier to mine frequent communication patterns, raising privacy concerns. However, our benchmarks are generated from public datasets and do not contain any sensitive information.

C RELATED WORK

Graph Representation Learning Graph neural networks (GNNs) are a powerful framework for learning representations of graph-structured data (Li et al., 2016; Gilmer et al., 2017; Kipf et al., 2017). They aggregate node and edge features through multiple rounds of message passing to obtain graph embeddings (Gilmer et al., 2017; Xu et al., 2019). GNNs have found notable success in problems related to node classification, graph classification and similarity search (Kipf et al., 2017; Hamilton et al., 2017; Xu et al., 2019; Bai et al., 2019).

Neural Subgraph Matching and Graph Retrieval Many recent works use GNNs to detect subgraph relations or retrieve graphs by approximating subgraph matching (Ying et al., 2020; Roy et al., 2022; Ramachandran et al., 2024). GEN (Li et al., 2019) independently maps query and corpus graphs to vector representations through message passing and graph-level pooling before computing their similarity score. IsoNet (Roy et al., 2022) retrieves graphs by learning a differentiable node or edge alignment for subgraph isomorphism.

Sequence Alignment and Subsequence Retrieval Classical dynamic-programming methods study symbolic sequence alignment (Needleman et al., 1970; Smith et al., 1981), while related time-series methods align real-valued sequences through dynamic time warping and soft-DTW (Sakoe et al., 1978; Cuturi et al., 2017). OPAS (Kohli et al., 2026) models order-preserving subsequence selection through a relaxed staircase-alignment formulation.

Differentiable assignment and optimal transport Certain graph retrieval models use soft doubly stochastic alignments to approximate injective node or edge correspondences (Roy et al., 2022; Ramachandran et al., 2024). They relax the underlying assignment problem to an entropy-regularized optimal transport problem with a learned cost matrix (Cuturi, 2013) and subsequently solve it by alternating row and column normalizations (Sinkhorn, 1967).

Chemical Reaction Modeling Chemical reactions can be modeled as sequences of graphs with nodes as atoms and edges as bonds. SynTemp (Phan et al., 2025) extracts and clusters reaction rules from reaction databases, while (Unsleber, 2023) extracts reaction templates which can be used to filter first-principles-based explorations.

Dynamic Graph Representation Learning Learning representations of dynamic graphs means encoding information about the graph's structure and its temporal evolution into graph representations. Different methods differ primarily in how they obtain structural and temporal information. DySAT (Sankar et al., 2020) applies attention over graph neighbourhoods and each node's representation history, while ROLAND (You et al., 2022) recurrently updates node representations at every GNN layer across successive snapshots. GraphSSM (Li et al., 2024b) combines graph-based feature aggregation with state-space updates to maintain a fixed-size memory of each node's evolving structural context.

D TECHNICAL RESULTS AND THEIR PROOFS

D.1 FACTORING THE STAIRCASE SELECTOR

In this section we prove that Eq. (5) and Eq. (6) are equivalent. Recall that Eq. (5) writes the SIGMA relevance distance as

$$\Delta(S_q, S_c) = \min_{\boldsymbol{P}\in\Pi_n, \boldsymbol{B}} \sum_{t\in[\ell_q]} \left\| \left[\boldsymbol{A}_{q,t} - \boldsymbol{P} \left(\sum_{\tau\in[\ell_c]} \boldsymbol{B}[t,\tau]\boldsymbol{A}_{c,\tau} \right) \boldsymbol{P}^\top \right]_+ \right\|_{1,1}, \text{ s.t. } b_1\mathbf{1} \le \mathbf{LBa} \le b_2\mathbf{1} \tag{16}$$

By construction, $\boldsymbol{B}[t,\mu(t)] = 1$ and $\boldsymbol{B}[t,\tau] = 0$ for all $\tau \neq \mu(t)$, where μ is the order preserving selection function. Thus

$$\left\| \left[\boldsymbol{A}_{q,t} - \boldsymbol{P} \left(\sum_{\tau\in[\ell_c]} \boldsymbol{B}[t,\tau]\boldsymbol{A}_{c,\tau} \right) \boldsymbol{P}^\top \right]_+ \right\|_{1,1} = \left\| \left[\boldsymbol{A}_{q,t} - \boldsymbol{P}\boldsymbol{A}_{c,\mu(t)}\boldsymbol{P}^\top \right]_+ \right\|_{1,1}. \tag{17}$$

Since row t of $\boldsymbol{B}$ is one-hot

$$\left\| \left[\boldsymbol{A}_{q,t} - \boldsymbol{P}\boldsymbol{A}_{c,\mu(t)}\boldsymbol{P}^\top \right]_+ \right\|_{1,1} = \sum_{\tau\in[\ell_c]} \boldsymbol{B}[t,\tau] \left\| \left[\boldsymbol{A}_{q,t} - \boldsymbol{P}\boldsymbol{A}_{c,\tau}\boldsymbol{P}^\top \right]_+ \right\|_{1,1}. \tag{18}$$

Finally, summing over $t \in [\ell_q]$ gives Eq. (6)

$$\Delta(S_q, S_c) = \min_{\boldsymbol{P}\in\Pi_n, \boldsymbol{B}} \sum_{t\in[\ell_q]} \sum_{\tau\in[\ell_c]} \boldsymbol{B}[t,\tau] \left\| \left[\boldsymbol{A}_{q,t} - \boldsymbol{P}\boldsymbol{A}_{c,\tau}\boldsymbol{P}^\top \right]_+ \right\|_{1,1}, \text{ s.t. } b_1\mathbf{1} \le \mathbf{LBa} \le b_2\mathbf{1} \tag{19}$$

D.2 REWARD MATRIX FOR SEQUENCE ALIGNMENT

Next, we show how to obtain Eq. (13) from Eq. (12).

Start by representing the staircase matrix $\boldsymbol{B}$ as $\boldsymbol{B} = \boldsymbol{M}\boldsymbol{R}$ where $\boldsymbol{M} \in \{0,1\}^{\ell_q \times \ell_c}$ selects the first ℓ_q rows and $\boldsymbol{R} \in \Pi_{\ell_c}$ is a permutation matrix. Fix the node alignment $\boldsymbol{P}$ and write $\mathrm{cost}(\boldsymbol{P})[t,\tau] = \left\| [\boldsymbol{Q}_t - \boldsymbol{P}\boldsymbol{C}_\tau]_+ \right\|_{1,1}$ as the graph-pair cost matrix.

The sequence-alignment objective in Eq. (12) is

$$J(\boldsymbol{R}) = \langle \boldsymbol{M}\boldsymbol{R}, \mathrm{cost}(\boldsymbol{P}) \rangle_F + \langle \boldsymbol{\lambda}_1, b_1\mathbf{1} - \boldsymbol{L}\boldsymbol{M}\boldsymbol{R}\boldsymbol{a} \rangle + \langle \boldsymbol{\lambda}_2, \boldsymbol{L}\boldsymbol{M}\boldsymbol{R}\boldsymbol{a} - b_2\mathbf{1} \rangle. \tag{20}$$

Here, $\boldsymbol{a} \in \mathbb{R}^{\ell_c}$ is the positional code (i.e., $\boldsymbol{a}[i] = i$ is a monotone function of i) and $\boldsymbol{L}$ is the fixed linear operator used to compute successive differences of the selected positions. The scalars b_1, b_2 specify the lower and upper interval targets, and $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2$ are the corresponding nonnegative multipliers.

Observe that

$$J'(\boldsymbol{R}) = \langle \boldsymbol{M}\boldsymbol{R}, \mathrm{cost}(\boldsymbol{P}) \rangle_F + \langle \boldsymbol{\lambda}_2 - \boldsymbol{\lambda}_1, \boldsymbol{L}\boldsymbol{M}\boldsymbol{R}\boldsymbol{a} \rangle \tag{21}$$

where we have dropped the $\boldsymbol{R}$-independent terms since they don't impact the minimizer. By the cyclic property of trace

$$\begin{aligned} \langle \boldsymbol{M}\boldsymbol{R}, \mathrm{cost}(\boldsymbol{P}) \rangle_F &= \mathrm{tr}\big((\boldsymbol{M}\boldsymbol{R})^\top \mathrm{cost}(\boldsymbol{P}) \big) && (22) \\ &= \mathrm{tr}\big(\boldsymbol{R}^\top \boldsymbol{M}^\top \mathrm{cost}(\boldsymbol{P}) \big) && (23) \\ &= \langle \boldsymbol{M}^\top \mathrm{cost}(\boldsymbol{P}), \boldsymbol{R} \rangle_F. && (24) \end{aligned}$$

and

$$\langle \boldsymbol{\lambda}_2 - \boldsymbol{\lambda}_1, \boldsymbol{L}\boldsymbol{M}\boldsymbol{R}\boldsymbol{a} \rangle = \operatorname{tr}\big((\boldsymbol{\lambda}_2 - \boldsymbol{\lambda}_1)^\top \boldsymbol{L}\boldsymbol{M}\boldsymbol{R}\boldsymbol{a}\big) \tag{25}$$

$$= \operatorname{tr}\big(\boldsymbol{R}^\top \boldsymbol{M}^\top \boldsymbol{L}^\top (\boldsymbol{\lambda}_2 - \boldsymbol{\lambda}_1)\boldsymbol{a}^\top\big) \tag{26}$$

$$= \big\langle \boldsymbol{M}^\top \boldsymbol{L}^\top (\boldsymbol{\lambda}_2 - \boldsymbol{\lambda}_1)\boldsymbol{a}^\top, \boldsymbol{R} \big\rangle_F . \tag{27}$$

Hence the objective in Eq.(21) can be expressed as

$$J'(\boldsymbol{R}) = \big\langle \boldsymbol{M}^\top \operatorname{cost}(\boldsymbol{P}) + \boldsymbol{M}^\top \boldsymbol{L}^\top (\boldsymbol{\lambda}_2 - \boldsymbol{\lambda}_1)\boldsymbol{a}^\top, \boldsymbol{R} \big\rangle_F \tag{28}$$

and the corresponding optimization problem reduces to

$$\begin{aligned} \boldsymbol{R}^\star \in \arg\min_{\boldsymbol{R}\in\Pi_{\ell_c}} J(\boldsymbol{R}) = \arg\max_{\boldsymbol{R}\in\Pi_{\ell_c}} \langle \boldsymbol{R}, \boldsymbol{\Omega} \rangle_F, \\ \text{where} \qquad \boldsymbol{\Omega} = \boldsymbol{M}^\top \left(\boldsymbol{L}^\top (\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_2)\boldsymbol{a}^\top - \operatorname{cost}(\boldsymbol{P}) \right). \end{aligned} \tag{29}$$

Which is just Eq. (13).

D.3 Sinkhorn relaxation of the sequence assignment

Eq. 29 is a linear assignment problem over permutation matrices. To obtain a differentiable approximation of the solution, we relax the feasible set to the Birkhoff polytope $\mathcal{B}_{\ell_c}$ and add entropic regularization (Mena et al., 2018)

$$\boldsymbol{R}_\tau = \arg\max_{\boldsymbol{R}\in\mathcal{B}_{\ell_c}} \langle \boldsymbol{R}, \boldsymbol{\Omega} \rangle_F + \tau h(\boldsymbol{R}), \qquad h(\boldsymbol{R}) = -\sum_{i,j} \boldsymbol{R}_{ij} \log \boldsymbol{R}_{ij}. \tag{30}$$

This can be solved by applying Sinkhorn normalization to the temperature-scaled reward matrix, $\boldsymbol{R}_\tau = \operatorname{Sinkhorn}(\boldsymbol{\Omega}/\tau)$ where τ is the temperature parameter. The relaxed staircase alignment of Section 4.3 is then given by $\boldsymbol{B}_\tau = \boldsymbol{M}\boldsymbol{R}_\tau$.

E Details about experimental setup

E.1 Datasets

We construct five graph-sequence retrieval benchmarks from PaRoutes, ICEWS, Ito, Uetz and FlowER.

Dataset	$\lvert Q\rvert$	$\lvert C\rvert$	Mean ℓ_q	Mean ℓ_c	Mean pos/query	Min pos/query	Max pos/query	pairs(1)	pairs(−1)	$\frac{\text{pairs}(1)}{\text{pairs}(-1)}$
PaRoutes	122	1800	3.16	6.54	232.47	151	390	28361	191239	0.148
ICEWS	300	800	3.00	20.00	176.22	80	319	52866	187134	0.283
Ito	300	800	4.87	18.98	171.60	80	320	51480	188520	0.273
Uetz	286	800	4.82	18.82	189.83	81	319	54292	174508	0.311
FlowER	204	800	3.50	7.30	112.78	65	174	23008	140192	0.164

Table 8: Sequence-level retrieval statistics for the generated benchmarks. $|Q|$ and $|C|$ denote the total numbers of query and corpus sequences, respectively. pos/query denotes the number of relevant corpus sequences for a query, and $\text{pairs}(y)$ denotes the number of query–corpus sequence pairs with label $y \in \{+1, -1\}$.

Each benchmark consists of a fixed corpus of graph sequences and a separate collection of query sequences. We partition the queries into training, validation, and test sets in proportions of 60%, 15%, and 25%, respectively, while keeping the corpus common to all three splits. Categorical node and edge attributes are encoded using one-hot vectors, while continuous measurements are retained as numerical features. Ground truth relevance is determined by a combinatorial solver (RDKit (RDK) for chemistry derived graphs, Glasgow solver (McCreesh et al., 2020) for the rest).

PaRoutes [1] (Genheden et al., 2022) The PaRoutes dataset consists of multistep retrosynthetic pathways for various targets. Since each target can have multiple associated pathways we build our corpus sequences by extracting the longest such pathway. Queries are constructed by following an atom involved in a bond order change and deduplicated to avoid redundancy. Ground truth labels are obtained via RDKit.

ICEWS [2] (Li et al., 2021; Boschee et al., 2015; Jin et al., 2020) ICEWS (Integrated Crisis Early Warning System) is a dataset consisting of interactions between sociopolitical actors. We use the ICEWS-18 dataset distributed by RE-GCN (Li et al., 2021), representing actors as nodes and their interactions as edges. The dataset is bucketed into three-day intervals and the first 80 timesteps are used for corpus construction. Corpus elements are constructed by choosing a 25-actor neighborhood from a node and following it across 20 timesteps. Queries are constructed analogously from the remaining 21 timesteps, but are smaller in size, following $6-7$ actors across three timesteps. Ground truth relevance labels are obtained using the Glasgow Subgraph Solver (McCreesh et al., 2020).

DPPIN (Uetz and Ito) [3] (Fu et al., 2022) DPPIN is a repository of 12 protein-protein interaction network datasets. We use two of these datasets, namely, Uetz and Ito, and run a sliding-window of size 3 across time to build graph sequences, obtaining 34 steps. The first 22 are reserved for corpus generation and the remaining for query generation. A corpus item is built from a node by tracking a 25-node neighborhood across 20 steps. Queries track $14-16$-node neighborhoods across $4-6$ steps. Ground truth relevance labels are obtained via the Glasgow Subgraph Solver (McCreesh et al., 2020).

FlowER [4] (Joung et al., 2025) The FlowER dataset consists of chemical reactions consisting of multiple elementary steps each, which we represent as graph sequences. Nodes carry element identity and formal charge as features, while edges carry bond order as features. We restrict our attention to reactions with at least 7 steps and at most 50 distinct atoms across the steps and select 800 valid sequences as our corpus. Queries are constructed from local neighborhoods of broken bonds and filtered for uniqueness such that no two queries are duplicates, nor is one contained in the other. We use RDKit as our combinatorial solver to obtain ground truth relevance labels for the dataset.

[1] https://doi.org/10.5281/zenodo.6275421
[2] https://github.com/Lee-zix/RE-GCN
[3] https://github.com/DongqiFu/DPPIN
[4] https://github.com/FongMunHong/FlowER

E.2 BASELINES FOR GRAPH-MATCHING

We compare SIGMANET against five neural graph-matching baselines and three temporal graph encoders adapted to graph-sequence retrieval.

(1) For GEN (Li et al., 2019), GREED (Ranjan et al., 2022), SimGNN (Bai et al., 2019), Neuromatch (Ying et al., 2020) and IsoNet (Roy et al., 2022) we use the respective reference implementations.

(2) For GraphSSM (Li et al., 2024b), MaskDGNN (He et al., 2025) and UnderGS (Wu et al., 2026) we use the official repositories as references, while adapting the implementations to our use case.

Implementation changes For GraphSSM, we: **(1)** freeze states of masked (inactive) nodes, **(2)** obtain node features without cross-channel state-mixing, and **(3)** increase number of GraphSSM blocks from 2 to 5. For MaskDGNN we apply frequency filtering per node across time and recompute masks at each forward pass. For UnderGS, we: **(1)** update Temporal Cohesive Neighbour Store (TCNS) neighbours of a node using its multi-hop neighbourhood, **(2)** update node states by adding the current features to the previous state, followed by message passing and a two-layer MLP, and **(3)** propagate the gradients through previous timestep embeddings.

Adaptation to graph-sequence retrieval Consider a query sequence $S_q = (G_{q,1}, \ldots, G_{q,\ell_q})$ and a corpus sequence $S_c = (G_{c,1}, \ldots, G_{c,\ell_c})$. Graph matching baselines directly give us a matching score for each graph-pair $(G_{q,t}, G_{c,\tau})$ as well as node-level and graph-level embeddings. We pass the matching scores to the SDTW matcher to obtain relevance score for sequence pairs. For temporal baselines we aggregate the per-step node level embeddings into graph level embeddings. We compute the pairwise matching score for each graph-pair as the asymmetric hinge distance between the graph-level embeddings, and pass this set of scores to the SDTW matcher to get the final sequence-level relevance scores.

SDTW Soft-DTW (Cuturi et al., 2017) is a smooth relaxation of Dynamic Time Warping (DTW). Given the graph-pair cost matrix cost, it computes a differentiable soft minimum over valid DTW alignment paths. This is the aligner used by our baselines.

E.3 ADDITIONAL DETAILS OF SIGMANET

GNN layer The GNN layer is a message-passing layer with initial embeddings obtained from a small MLP acting on raw node and edge features. For each edge, messages are computed by a linear layer applied to a concatenation of the endpoint's embeddings and the edge's embedding. Messages associated to each node's incoming edges are summed and the node embedding is updated using a Gated Recurrent Unit (GRU). In all, we use a total of $K = 5$ propagation layers with temporal contextualization interleaved in GSEQENC.

Interval-penalty multipliers The GSEQENC module outputs embeddings after passing them through a causal transformer. Hence, for each query and corpus timestep, we directly apply attention-based node pooling,

$$\widetilde{\boldsymbol{q}}_t = \mathrm{Pool}_{\mathrm{attn}}(\boldsymbol{Q}_t), \qquad \widetilde{\boldsymbol{c}}_\tau = \mathrm{Pool}_{\mathrm{attn}}(\boldsymbol{C}_\tau),$$

and stack the resulting vectors into $\widetilde{\boldsymbol{Q}} \in \mathbb{R}^{\ell_q \times d}$ and $\widetilde{\boldsymbol{C}} \in \mathbb{R}^{\ell_c \times d}$. Next we form the timestep-level similarity representation

$$\psi(\boldsymbol{Q}, \boldsymbol{C}) = \widetilde{\boldsymbol{Q}}\widetilde{\boldsymbol{C}}^\top \in \mathbb{R}^{\ell_q \times \ell_c}.$$

and use two separate multiplier networks

$$\boldsymbol{\lambda}_1 = \mathrm{LagNet}_\phi^{\mathrm{low}}(\psi(\boldsymbol{Q}, \boldsymbol{C}))\,, \qquad \boldsymbol{\lambda}_2 = \mathrm{LagNet}_\phi^{\mathrm{up}}(\psi(\boldsymbol{Q}, \boldsymbol{C}))\,, \qquad \boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2 \in \mathbb{R}_{\geq 0}^{\ell_q - 1}.$$

to parameterize the lower- and upper-threshold interval penalties used by the final scorer, respectively. Note that for our experiments, since we set $b_2 \to \infty$, $\mathrm{LagNet}_\phi^{\mathrm{up}}$ was inactive.

$\mathbf{LagNet}_\phi$ LagNet_ϕ is a convolutional neural network that processes its input similarity matrix with convolutions along the corpus dimension. The output is adaptively pooled across the corpus axis, followed by an MLP and sigmoid nonlinearity in order to obtain the corresponding multiplier $\boldsymbol{\lambda}$.

Relevance score Let $\mathrm{cost}(\boldsymbol{P}) \in \mathbb{R}^{\ell_q \times \ell_c}$ denote the neural graph-pair cost matrix induced by the shared node alignment $\boldsymbol{P}$, and let $\boldsymbol{B} = \boldsymbol{M}\boldsymbol{R}$ be the Sinkhorn-relaxed staircase selector. We define the matching score as

$$s_{\mathrm{match}}(S_q, S_c) = -\langle \boldsymbol{B}, \mathrm{cost}(\boldsymbol{P}) \rangle_F \tag{31}$$

and the lower and upper order violation scores respectively as

$$s_{\text{low}}(S_q, S_c) = -\left\langle \boldsymbol{\lambda}_1, [b_1 \mathbf{1} - \boldsymbol{LBa}]_+ \right\rangle, \tag{32}$$

$$s_{\text{up}}(S_q, S_c) = -\left\langle \boldsymbol{\lambda}_2, [\boldsymbol{LBa} - b_2 \mathbf{1}]_+ \right\rangle. \tag{33}$$

The final relevance score is obtained by a learned linear combination of these scores

$$\text{score}(S_q, S_c) = \beta_1 \, s_{\text{match}} + \beta_2 \, s_{\text{low}} + \beta_3 \, s_{\text{up}} \qquad \beta_1, \beta_2, \beta_3 > 0. \tag{34}$$

where β_1, β_2 and β_3 are positive learnable scalars.

Training objective We train SIGMANET with the pairwise hinge-ranking objective

$$\mathcal{L} = \sum_{q \in Q} \sum_{c^+ \in C_{\checkmark}(q)} \sum_{c^- \in C_{\times}(q)} \Big[\Delta + s(S_q, S_{c^-}) - s(S_q, S_{c^+}) \Big]_+ \tag{35}$$

where, for a query q, $C_{\checkmark}(q)$ and $C_{\times}(q)$ denote the relevant and irrelevant corpus items respectively.

E.4 TRAINING PROTOCOL AND EVALUATION METRICS

All checkpoints are selected using validation MAP, but we report both test MAP and test MRR for all our experiments.

Mean Average Precision For a query q, consider the ordered sequence of corpus items with decreasing relevance score. Let $\text{rel}_q(k) \in \{0, 1\}$ denote whether the corpus item at rank k is relevant. This allows us to define the precision at rank k as $P_q(k) = \frac{1}{k} \sum_{i=1}^{k} \text{rel}_q(i)$. The average precision of a query q is then

$$\text{AP}(q) = \frac{1}{|C_{\checkmark}(q)|} \sum_{k=1}^{|C|} P_q(k) \text{rel}_q(k)$$

Averaging this over the whole query set gives the Mean Average Precision (MAP)

$$\text{MAP} = \frac{1}{|Q|} \sum_{q \in Q} \text{AP}(q)$$

MAP is our primary metric because each query can have many relevant corpus items, and it rewards rankers that place all relevant items towards the top of the ranked list.

Mean Reciprocal Rank Let $r_q^\star$ be the rank of the highest-ranked relevant corpus item for query q. The Reciprocal Rank (RR) for q is

$$RR(q) = \frac{1}{r_q^\star},$$

and the Mean Reciprocal Rank (MRR) is

$$MRR = \frac{1}{|Q|} \sum_{q \in Q} RR(q).$$

MRR is a secondary metric for us as it can complement MAP by measuring first-hit quality, but by itself does not measure how well any other relevant items may be scored.

E.5 HYPERPARAMETERS

Unless stated otherwise, all models are trained with Adam, learning rate 5×10^{-4}, weight decay 10^{-5}, batch size 1024, and no gradient accumulation. We evaluate once per epoch and select checkpoints using validation MAP. Early stopping uses patience 100 and tolerance 10^{-4}, with a maximum of $10{,}000$ epochs.

Constants related to sequence-matchers Sinkhorn normalization uses 20 iterations, temperature 0.1, and no Gumbel noise. For *SDTW*, the smoothing parameter is $\gamma = 1.0$. The lower-gap threshold is $b_1 = 1$. Our experiments use $b_2 \to \infty$, allowing arbitrary gaps between successive matched corpus timesteps.

Temporal contextualizer TF_θ The causal decoder TF_θ uses $d_{\text{model}} = 16$, 2 attention heads, 1 transformer layer, feed-forward width 24, and dropout 0.1. Its input and output projections operate on 10-dimensional node states. The post-processing causal transformer TF'_θ in SIGMANET uses $d_{\text{model}} = 32$, 4 attention heads, 1 transformer layer, feed-forward width 64, and dropout 0.1. Both transformers use learned temporal positional embeddings, with maximum sequence length 14 for

FlowER and PaRoutes and 20 for Uetz, Ito, and ICEWS. Unless stated otherwise, the TF_θ is used in the *PRE+Skip* configuration.

E.6 Software and hardware

All experiments are implemented using PyTorch, and run on NVIDIA RTX A6000 GPUs with 48 GB memory. To balance throughput and memory headroom, we typically spawned three runs concurrently on a single A6000 and tracked them using `wandb`.

E.7 Trainable parameters

Parameters in baselines Table 9 reports parameter counts for the baseline encoders.

Encoder	PaRoutes	ICEWS	Ito	Uetz	FlowER
GEN	3,170	1,930	1,750	1,750	3,150
GREED	3,200	1,940	1,840	1,840	3,180
SimGNN	1,700	440	340	340	1,680
Neuromatch	4,823	3,563	3,463	3,463	4,803
IsoNet	2,840	1,600	1,420	1,420	2,820
GraphSSM	6,185	4,925	4,825	4,825	6,165
MaskDGNN	2,250	990	890	890	2,230
UnderGS	2,370	1,120	980	980	2,350

Table 9: Trainable parameter counts of graph and temporal graph encoders. These counts do not include any sequence alignment or scoring head parameters.

Component-wise decomposition of SigmaNet Table 10 reports the complete parameter decomposition of SigmaNet for each benchmark where $b_2 \to \infty$.

Module	PaRoutes	ICEWS	Ito	Uetz	FlowER
$\mathrm{GSEQENC}_\theta$ excluding TF'_θ	5,612	4,468	4,288	4,288	5,592
Post-processing contextualizer TF'_θ	9,408	9,600	9,600	9,600	9,408
Node-alignment projection network (LRL_θ)	800	800	800	800	800
Attention-based node pooling ($\mathrm{Pool}_{\mathrm{attn}}$)	33	33	33	33	33
Lagrangian multiplier network ($\mathrm{LagNet}^{\mathrm{low}}_\phi$)	298	298	298	298	298
Scoring coefficients (β_1, β_2)	2	2	2	2	2
Total	16,153	15,201	15,021	15,021	16,133

Table 10: Component-wise trainable parameter counts of SigmaNet.

F Additional Experimental Results

F.1 Ablation Study: Effect of Temporal Contextualizer TF_θ

To assess the effect of temporal contextualization, we add TF_θ to baselines and remove it from SigmaNet. The baselines use SDTW as sequence matcher. Tables 11 and 12 summarize the MAP and MRR results. We make the following observations. **(1)** Adding TF_θ to SigmaNet improves MAP on four out of five datasets and MRR on all five. The only exception is Ito, where MAP decreases slightly. **(2)** TF_θ usually improves the performance on the baselines with the exception of IsoNet.

Baselines	PaRoutes	ICEWS	Ito	Uetz	FlowER
GEN	0.757	0.478	0.753	0.841	0.699
GEN + TF_θ	0.721	0.471	0.849	0.921	0.679
Neuromatch	0.714	0.486	0.693	0.801	0.688
Neuromatch + TF_θ	0.732	0.484	0.707	0.826	0.609
IsoNet	0.761	<u>0.513</u>	0.838	<u>0.936</u>	0.793
IsoNet + TF_θ	0.732	0.505	0.824	0.879	<u>0.796</u>
GraphSSM	0.694	0.461	0.735	0.862	0.761
GraphSSM + TF_θ	0.705	0.488	0.745	0.886	0.686
MaskDGNN	0.586	0.456	0.732	0.829	0.648
MaskDGNN + TF_θ	0.606	0.456	0.758	0.852	0.640
UnderGS	0.573	0.437	0.700	0.782	0.667
UnderGS + TF_θ	0.639	0.450	0.767	0.874	0.607
SigmaNet - TF_θ	<u>0.807</u>	0.489	**0.853**	0.928	0.792
SigmaNet	**0.818**	**0.527**	<u>0.851</u>	**0.948**	**0.812**

Table 11: MAP performance with and without TF_θ for SigmaNet and the baselines. Adding TF_θ lowers MAP on FlowER for every baseline shown except IsoNet. Adding TF_θ improves Neuromatch and MaskDGNN on most datasets. **Bold** denotes the best and <u>underline</u> denotes the second-best value in each column.

Baselines	PaRoutes	ICEWS	Ito	Uetz	FlowER
GEN	0.866	0.693	0.976	0.955	0.872
GEN + TF_θ	0.786	0.705	**1.000**	<u>0.982</u>	0.840
Neuromatch	0.766	0.711	0.970	0.908	0.871
Neuromatch + TF_θ	0.857	0.709	0.937	0.919	0.722
IsoNet	0.865	0.678	0.987	**1.000**	**0.915**
IsoNet + TF_θ	0.770	0.702	0.984	0.976	<u>0.908</u>
GraphSSM	0.800	0.675	0.923	0.939	0.886
GraphSSM + TF_θ	0.801	0.696	0.970	0.886	0.860
MaskDGNN	0.692	0.721	0.956	0.889	0.709
MaskDGNN + TF_θ	<u>0.913</u>	0.646	0.984	0.910	0.799
UnderGS	0.745	<u>0.726</u>	0.922	0.870	0.783
UnderGS + TF_θ	0.773	0.625	0.971	0.961	0.758
SigmaNet - TF_θ	0.896	0.692	0.987	0.973	0.893
SigmaNet	**0.938**	**0.783**	<u>0.991</u>	0.979	**0.915**

Table 12: MRR performance with and without TF_θ for SigmaNet and the baselines. Adding TF_θ improves MRR for SigmaNet on every dataset. GEN + TF_θ achieves highest MRR on Ito and IsoNet on Uetz. **Bold** denotes the best and <u>underline</u> denotes the second-best value in each column.

F.2 Ablation Study: Effect of Staircase aligner (SA)

To ablate the effect of Staircase aligner (SA), we compare SigmaNet and the baselines with and without SA as the sequence matcher. Tables 13 and 14 summarize the MAP and MRR results. We make the following observations. **(1)** Removing SA lowers MAP and MRR of SigmaNet on every dataset. **(2)** Adding SA improves baseline performance in most of the cases. In particular, MAP performance of Neuromatch, MaskDGNN, and UnderGS is improved on all datasets. **(3)** The gains from Staircase aligner suggest that a learned order-preserving timestep alignment is more effective than post-hoc sequence aggregation.

Baselines	PaRoutes	ICEWS	Ito	Uetz	FlowER
GEN	0.757	0.478	0.753	0.841	0.699
GEN + SA	**0.822**	0.482	0.835	0.917	0.694
Neuromatch	0.714	0.486	0.693	0.801	0.688
Neuromatch + SA	0.775	0.491	0.758	0.866	0.738
IsoNet	0.761	<u>0.513</u>	0.838	<u>0.936</u>	0.793
IsoNet + SA	0.807	0.489	**0.853**	0.928	0.792
GraphSSM	0.694	0.461	0.735	0.862	0.761
GraphSSM + SA	0.744	0.495	0.729	0.886	**0.814**
MaskDGNN	0.586	0.456	0.732	0.829	0.648
MaskDGNN + SA	0.745	0.500	0.762	0.874	0.797
UnderGS	0.573	0.437	0.700	0.782	0.667
UnderGS + SA	0.727	0.467	0.772	0.888	0.760
SigmaNet - SA	0.732	0.505	0.824	0.879	0.796
SigmaNet	<u>0.818</u>	**0.527**	<u>0.851</u>	**0.948**	<u>0.812</u>

Table 13: MAP performance with and without Staircase aligner (SA) as the sequence matcher for SigmaNet and the baselines. Adding SA improves MAP in most baseline comparisons. Adding SA improves Neuromatch, MaskDGNN and UnderGS on all the datasets. **Bold** denotes the best and <u>underline</u> denotes the second-best value in each column.

Baselines	PaRoutes	ICEWS	Ito	Uetz	FlowER
GEN	0.866	0.693	0.976	0.955	0.872
GEN + SA	**0.939**	0.718	0.962	0.974	0.838
Neuromatch	0.766	0.711	0.970	0.908	0.871
Neuromatch + SA	0.851	0.654	0.980	0.803	0.809
IsoNet	0.865	0.678	<u>0.987</u>	**1.000**	**0.915**
IsoNet + SA	0.896	0.692	<u>0.987</u>	0.973	0.893
GraphSSM	0.800	0.675	0.923	0.939	0.886
GraphSSM + SA	0.830	0.696	0.983	0.940	0.881
MaskDGNN	0.692	0.721	0.956	0.889	0.709
MaskDGNN + SA	0.822	<u>0.728</u>	0.888	0.960	0.852
UnderGS	0.745	0.726	0.922	0.870	0.783
UnderGS + SA	0.821	0.681	0.901	0.934	0.895
SigmaNet - SA	0.770	0.702	0.984	0.976	<u>0.908</u>
SigmaNet	<u>0.938</u>	**0.783**	**0.991**	<u>0.979</u>	**0.915**

Table 14: MRR performance with and without Staircase aligner (SA) as the sequence matcher for SigmaNet and the baselines. Adding SA improves SigmaNet's MRR on all five datasets. Among the methods shown, SigmaNet leads on three datasets whereas GEN + SA is marginally better on PaRoutes and IsoNet on Uetz. **Bold** denotes the best and <u>underline</u> denotes the second-best value in each column.

F.3 Statistical Significance of Retrieval Results

We perform a statistical significance test on our results using a paired t-test on the MAP gain our method achieves over baselines. We show the p-values on the test set for 10 splits in Table 15. We observe that except for one case, the performance gain is statistically significant with $p < 0.05$.

Baselines	PaRoutes	ICEWS	Ito	Uetz	FlowER
GEN	**2.21e-6**	**4.21e-4**	**6.54e-9**	**1.03e-11**	**2.02e-9**
GREED	**1.76e-6**	**2.08e-4**	**3.22e-10**	**5.54e-11**	**1.00e-10**
SimGNN	**4.21e-7**	**1.87e-6**	**1.23e-10**	**1.25e-12**	**5.60e-11**
Neuromatch	**8.84e-7**	**0.0017**	**7.85e-10**	**2.58e-10**	**3.07e-9**
IsoNet	**4.29e-6**	0.2434	**0.0021**	**0.0049**	**8.83e-5**
GraphSSM	**4.83e-7**	**1.73e-6**	**3.21e-9**	**1.39e-10**	**1.87e-7**
MaskDGNN	**1.41e-8**	**8.03e-7**	**1.38e-8**	**3.20e-10**	**4.70e-9**
UnderGS	**2.59e-9**	**7.86e-7**	**9.53e-10**	**5.13e-10**	**4.65e-9**

Table 15: One-sided p-values from paired t-tests comparing SigmaNet with baselines using the *SDTW* sequence matcher. **Bold** indicates $p < 0.05$.

F.4 Ablation Study: Effect of Order Penalties

We examine the effect of the learned order penalties by comparing SigmaNet with a variant that removes the $\boldsymbol{\lambda}$-dependent terms, and scores each sequence pair only using the matching cost under the relaxed selector $\boldsymbol{B}$. Tables 16 and 17 summarize the MAP and MRR results, respectively. We make the following observations. **(1)** Removing the order penalty reduces the MAP on all five datasets and MRR on four datasets. **(2)** Without the learned penalty, the sequence matching is driven by graph-pair matching costs alone and hence, does not explicitly favor an order-preserving subsequence.

Methods	PaRoutes	ICEWS	Ito	Uetz	FlowER
SigmaNet (No $\boldsymbol{\lambda}$)	0.133	0.458	0.519	0.593	0.144
SigmaNet	**0.818**	**0.527**	**0.851**	**0.948**	**0.812**

Table 16: MAP performance of SigmaNet with and without the learned order penalty. Removing it lowers MAP on every dataset, with the largest drops on PaRoutes and FlowER. **Bold** denotes the best value in each column.

Methods	PaRoutes	ICEWS	Ito	Uetz	FlowER
SigmaNet (No $\boldsymbol{\lambda}$)	0.473	0.626	**0.993**	0.966	0.145
SigmaNet	**0.938**	**0.783**	0.991	**0.979**	**0.915**

Table 17: MRR performance of SigmaNet with and without the learned order penalty. Removing it lowers MRR on four datasets, while it improves slightly on Ito. **Bold** denotes the best value in each column.

F.5 Comparison of distinct-$\boldsymbol{P}_{t,\tau}$ with shared-$\boldsymbol{P}$

SigmaNet uses a single shared node alignment matrix $\boldsymbol{P}$. We now consider the case in which each graph pair (t, τ) has its own node alignment $\boldsymbol{P}_{t,\tau}$.

Relevance distance for distinct-$\boldsymbol{P}_{t,\tau}$ Define

$$d_{t,\tau}(\boldsymbol{P}) = \left\| \left[\boldsymbol{A}_{q,t} - \boldsymbol{P} \boldsymbol{A}_{c,\tau} \boldsymbol{P}^\top \right]_+ \right\|_{1,1}.$$

This allows us to write the distance between query and corpus sequences (Eq. 6) as

$$\Delta(S_q, S_c) = \min_{\boldsymbol{B}} \min_{\boldsymbol{P} \in \Pi_n} \sum_{t,\tau} \boldsymbol{B}[t,\tau]\, d_{t,\tau}(\boldsymbol{P}).$$

If each (t, τ) pair can have a distinct node alignment $\boldsymbol{P}_{t,\tau}$ the corresponding distance $\Delta_{\text{distinct}}(S_q, S_c)$ takes the form

$$\Delta_{\text{distinct}}(S_q, S_c) = \min_{\boldsymbol{B}} \sum_{t,\tau} \boldsymbol{B}[t,\tau] \min_{\boldsymbol{P}_{t,\tau} \in \Pi_n} d_{t,\tau}(\boldsymbol{P}_{t,\tau})$$

Neural Surrogate We estimate a distinct soft assignment $\boldsymbol{P}_{t,\tau} \in \mathcal{B}_n$ for each timestep pair (t, τ). The reward matrix in Eq. (13) subsequently takes the form

$$\boldsymbol{\Omega}_{\text{distinct}} = \boldsymbol{M}^\top \left(\boldsymbol{L}^\top (\boldsymbol{\lambda}_1 - \boldsymbol{\lambda}_2) \boldsymbol{a}^\top - \text{cost}_{\text{distinct}} \right)$$

where

$$\text{cost}_{\text{distinct}}[t,\tau] = \left\| [\boldsymbol{Q}_t - \boldsymbol{P}_{t,\tau} \boldsymbol{C}_\tau]_+ \right\|_{1,1}$$

is the graph-pair matching cost.

For completeness, Tables 19 and 20 reproduce the MAP comparison in Table 6 and report the corresponding MRR results. Table 18 reports peak GPU memory usage on PaRoutes. We make the following observations. **(1)** SigmaNet with shared $\boldsymbol{P}$ achieves higher MAP on every dataset and higher MRR on all but Uetz. **(2)** Although distinct $\boldsymbol{P}_{t,\tau}$ allows more flexibility by choosing a separate node alignment for every graph pair, this does not seem to improve performance. The results suggest that shared node alignment provides a useful inductive bias in applications where node identities are persistent. **(3)** Additionally, distinct node alignments mean we have to compute $\ell_q \times \ell_c$ many soft permutation matrices, thus increasing the memory footprint.

Methods	Training peak (GiB) ↓	Inference peak (GiB) ↓
Distinct $\boldsymbol{P}$	12.231	3.344
Shared $\boldsymbol{P}$	**6.797**	**1.876**

Table 18: Peak GPU memory during training and inference for SigmaNet with a shared node alignment $\boldsymbol{P}$ or distinct alignments $\boldsymbol{P}_{t,\tau}$ on PaRoutes. The shared-alignment setup uses less memory in both stages. **Bold** denotes the lower value in each column.

Methods	PaRoutes	ICEWS	Ito	Uetz	FlowER
Distinct $\boldsymbol{P}$	0.814	0.503	0.848	0.943	0.779
Shared $\boldsymbol{P}$	**0.818**	**0.527**	**0.851**	**0.948**	**0.812**

Table 19: Comparison with SigmaNet on test MAP after replacing the shared node-alignment matrix $\boldsymbol{P}$ with a distinct node-alignment matrix $\boldsymbol{P}_{t,\tau}$ for each graph pair. The shared alignment gives higher MAP on all five datasets. **Bold** denotes the best value in each column.

Methods	PaRoutes	ICEWS	Ito	Uetz	FlowER
Distinct $\boldsymbol{P}$	0.873	0.681	0.984	**0.991**	0.883
Shared $\boldsymbol{P}$	**0.938**	**0.783**	**0.991**	0.979	**0.915**

Table 20: Comparison with SigmaNet on test MRR after replacing the shared node-alignment matrix $\boldsymbol{P}$ with a distinct node-alignment matrix $\boldsymbol{P}_{t,\tau}$ for each graph pair. The shared alignment leads on four datasets except for Uetz. **Bold** denotes the best value in each column.

F.6 Comparison with Strict-DP Matcher

We compare Staircase aligner with a Dynamic Programming (DP) matching algorithm for finding an order-preserving subsequence. Given the graph-pair matching cost

$$c_{t,\tau} = \text{cost}(\boldsymbol{P})[t,\tau] = \left\|[\boldsymbol{Q}_t - \boldsymbol{P}\boldsymbol{C}_\tau]_+\right\|_{1,1}$$

and defining $D[t,\tau]$ as the minimum cost of matching the first t query graphs to a subsequence of the first τ corpus graphs, the DP recurrence is

$$D[t,\tau] = \min\left\{D[t,\tau-1], D[t-1,\tau-1] + c_{t,\tau}\right\} \quad \forall \quad 1 \le t \le \ell_q, 1 \le \tau \le \ell_c$$

with boundary conditions $D[0,\tau] = 0 \; \forall \; 0 \le \tau \le \ell_c$ and $D[t,0] = +\infty \; \forall \; 1 \le t \le \ell_q$. The corresponding score function is

$$s_{\text{DP}} = -D[\ell_q, \ell_c]$$

For training we use a differentiable relaxation of this problem, given by the recurrence

$$D_\gamma[t,\tau] = \text{softmin}_\gamma\left\{D_\gamma[t,\tau-1], D_\gamma[t-1,\tau-1] + c_{t,\tau}\right\}$$

where

$$\text{softmin}_\gamma(x,y) = -\gamma \log\left(e^{-x/\gamma} + e^{-y/\gamma}\right)$$

and γ $(= 1)$ is the softmin temperature. The corresponding boundary conditions remain the same, and the score is analogously

$$s_{\text{softDP}} = -D_\gamma[\ell_q, \ell_c].$$

This relaxation allows gradients to propagate through for end-to-end learning from relevance labels. Inference uses the original "hard" DP as described above. Tables 21 and 22 summarize the MAP and MRR results. We make the following observations.

(1) Staircase aligner achieves higher MAP and MRR than strict-DP on all five datasets, with significantly large gains on Uetz and FlowER. **(2)** Strict-DP finds the order-preserving subsequence with minimum cost using the learned graph-pair costs. However, it scores a sequence by simply accumulating these pairwise matching costs. The improvement with Staircase aligner over *SDTW* and DP suggests that its learned sequence alignment is better suited than post-hoc sequence aggregators.

Matchers	PaRoutes	ICEWS	Ito	Uetz	FlowER
Strict-DP	0.800	0.452	0.774	0.715	0.234
Staircase aligner	**0.818**	**0.527**	**0.851**	**0.948**	**0.812**

Table 21: MAP performance of SigmaNet using Staircase aligner and strict-DP as sequence matchers. Staircase aligner gives higher MAP on all the datasets. **Bold** denotes the best value in each column.

Matchers	PaRoutes	ICEWS	Ito	Uetz	FlowER
Strict-DP	0.920	0.638	0.981	0.845	0.367
Staircase aligner	**0.938**	**0.783**	**0.991**	**0.979**	**0.915**

Table 22: MRR performance of SigmaNet using Staircase aligner and strict-DP as sequence matchers. Staircase aligner gives higher MRR on all the datasets. **Bold** denotes the best value in each column.

F.7 Effect of Temporal Compactness: Finite b_2

We now consider the setting where b_2 is finite. In many applications, we are interested in capturing temporal evolution over a short time window, motivating us to favor temporally compact matched subsequences. We therefore set $b_2 = 2$, which penalizes alignments in which consecutive matched graphs are more than $b_2 = 2$ timesteps apart. Tables 23 and 24 summarize the MAP and MRR results, respectively. We make the following observations.

(1) MAP improves slightly with $b_2 = 2$ on PaRoutes, Uetz, and FlowER, but decreases on ICEWS and Ito. MRR decreases on all datasets except FlowER. The largest drop in both MAP and MRR is observed on ICEWS. **(2)** The mean gap between successive matched timesteps is 4.5 on ICEWS, compared to around 2.0 on the other datasets. This helps explain the larger performance drop on ICEWS, since setting $b_2 = 2$ penalizes the longer temporal gaps that are more common in this dataset. **(3)** Conversely, when the bias induced by finite b_2 aligns with temporal characteristics of datasets, the finite-b_2 runs can perform better.

Configuration	PaRoutes	ICEWS	Ito	Uetz	FlowER
SigmaNet ($b_2 = 2$)	**0.833**	0.363	0.841	**0.953**	**0.837**
SigmaNet	0.818	**0.527**	**0.851**	0.948	0.812

Table 23: MAP performance of SigmaNet with and without a finite upper bound ($b_2 = 2$ and $b_2 \to \infty$). Finite b_2 improves MAP on three of the five datasets. Both use $b_1 = 1$. **Bold** denotes the best value in each column.

Configuration	PaRoutes	ICEWS	Ito	Uetz	FlowER
SigmaNet ($b_2 = 2$)	0.915	0.539	0.963	0.977	**0.924**
SigmaNet	**0.938**	**0.783**	**0.991**	**0.979**	0.915

Table 24: MRR performance of SigmaNet with and without a finite upper bound ($b_2 = 2$ and $b_2 \to \infty$, respectively). Both use $b_1 = 1$. The $b_2 \to \infty$ setting leads on four out of five datasets. **Bold** denotes the best value in each column.

F.8 Runtime and Resource Usage

We report average training time per epoch and average inference time to rank the corpus for one query. Times were measured in terms of wall-clock time on NVIDIA RTX A6000 GPUs.

Training time per epoch Table 25 reports training time per epoch for SigmaNet and the baselines.

Baselines	PaRoutes	ICEWS	Ito	Uetz	FlowER
GEN	62.185	94.316	285.945	80.561	67.285
GREED	43.642	28.708	108.269	59.658	34.965
SimGNN	55.007	19.573	95.067	109.469	35.790
Neuromatch	42.954	27.387	97.106	59.174	29.085
IsoNet	60.454	46.129	88.119	87.264	42.783
GraphSSM	156.294	68.672	311.252	350.623	101.650
MaskDGNN	144.037	29.650	269.912	154.440	69.114
UnderGS	135.726	34.902	249.692	155.508	54.072
SigmaNet	287.536	116.090	377.476	448.642	162.611

Table 25: Average model training time (in seconds) per epoch over three measured epochs.

Inference time per query Table 26 reports forward pass time averaged over all held-out queries. Each query is scored against the complete corpus.

Together, these tables show that while SigmaNet is slower in terms of training time per epoch compared to baselines, once trained, inference time for SigmaNet is comparable to the baselines.

Baselines	PaRoutes	ICEWS	Ito	Uetz	FlowER
GEN	1142.321	104.019	964.778	942.563	178.122
GREED	504.494	134.589	624.467	340.151	179.292
SimGNN	324.344	69.728	112.501	282.721	109.876
Neuromatch	497.262	139.120	402.427	318.772	184.671
IsoNet	236.437	125.737	118.773	119.012	117.963
GraphSSM	316.444	120.057	271.088	163.123	162.046
MaskDGNN	211.765	73.071	151.208	121.342	88.243
UnderGS	494.588	132.292	282.142	234.693	297.944
SIGMANET	383.187	273.941	220.278	217.890	212.646

Table 26: Average forward pass time in milliseconds per query over the complete held-out query set.

Table 27 gives the average inference time for the interleaved graph encoder GSEQENC, TF'_θ, and the Sinkhorn modules for $\boldsymbol{P}$ and $\boldsymbol{R}$. We observe that the primary inference expense comes from the GSEQENC encoder.

Component	PaRoutes	ICEWS	Ito	Uetz	FlowER
GSEQENC excluding TF'_θ	324.643	236.352	177.928	175.745	176.872
Post-processing transformer TF'_θ	37.796	21.284	22.282	22.017	19.024
Sinkhorn for $\boldsymbol{P}$	4.919	4.849	4.607	4.709	4.661
Sinkhorn for $\boldsymbol{R}$	4.578	4.638	4.460	4.576	4.728

Table 27: Average component times of SIGMANET in milliseconds per query, measured in a separate synchronized pass over the complete held-out query set.

Runtime comparison against Exact Solvers. Table 28 compares the runtime of SIGMANET against the exact solver to solve the combinatorial optimization problem of SIGMA. We see that using an exact solver takes two orders of magnitude more time than SIGMANET, illustrating its utility for scalable retrieval.

Method	PaRoutes	ICEWS	Ito	Uetz	FlowER
SIGMANET	0.383	0.274	0.220	0.218	0.213
Exact solver	50.594	95.656	96.857	86.711	19.550

Table 28: Mean wall-clock inference time per query in seconds.