GraphMatch: Fusing Language and Graph Representations in a Dynamic Two-Sided Work Marketplace

Mikołaj Sacha[1*] **Hammad Jafri**[1] **Mattie Terzolo**[1] **Ayan Sinha**[1] **Andrew Rabinovich**[1]
[1]Upwork Inc.
[*]mikolajsacha@cloud.upwork.com

Abstract

Recommending matches in a text-rich, dynamic two-sided marketplace presents unique challenges due to evolving content and interaction graphs. We introduce GraphMatch, a new large-scale recommendation framework that fuses pre-trained language models with graph neural networks to overcome these challenges. Unlike prior approaches centered on standalone models, GraphMatch is a comprehensive recipe built on powerful text encoders and GNNs working in tandem. It employs adversarial negative sampling alongside point-in-time subgraph training to learn representations that capture both the fine-grained semantics of evolving text and the time-sensitive structure of the graph. We evaluated extensively on interaction data from Upwork, a leading labor marketplace, at large scale, and discuss our approach towards low-latency inference suitable for real-time use. In our experiments, GraphMatch outperforms language-only and graph-only baselines on matching tasks while being efficient at runtime. These results demonstrate that unifying language and graph representations yields a highly effective solution to text-rich, dynamic two-sided recommendations, bridging the gap between powerful pretrained LMs and large-scale graphs in practice.

1 Introduction

Real-world recommendation systems, from e-commerce platforms and job or talent marketplaces to academic citation networks and news feeds, can be naturally described as *Temporal Text-Attributed Graphs (TTAGs)*. In such TTAGs, nodes possess rich textual descriptions that evolve over time, while edges capture interactions like purchases, endorsements, or replies. Learning representations that effectively integrate *both* the fine-grained semantics of text and the global, time-dependent structure of the graph is crucial for tasks such as matching, recommendation, and retrieval. This remains a significant open research challenge.

Recent advances in contrastive pretraining have led to language models (LM) such as Sentence-BERT [28] and E5 [39]. These models distill documents into single embeddings and excel at measuring *local* semantic similarity. However, two important signals are usually missing when LMs are applied to TTAGs. First, the decision to link two items often depends on the neighborhood they inhabit within the graph, an aspect that pure text encoders cannot see. Secondly, in TTAGs, the meaning of a node and the evidence for linking can change rapidly. For e.g., the rise of generative AI transformed "AI Writing" jobs posts suggesting editorial skills to prompt engineering ones. Furthermore, geopolitical events continuously influence the freelancer-client matching dynamics in the global work marketplace. Treating text snapshots as static features risks serious information leakage from future states.

Preprint.

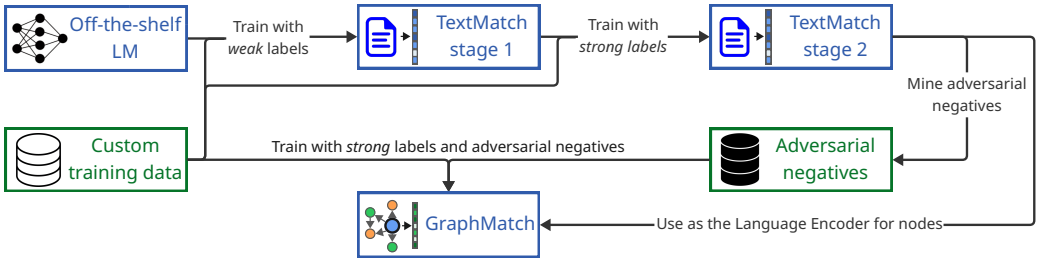

Figure 1: Flowchart of the multi-stage training of TextMatch and GraphMatch, with models and datasets at different stages.

Why graph neural networks alone are not enough? Graph neural networks (GNNs) and their temporal variants are designed to capture long-range dependencies and temporal patterns. However, applying them directly to large-scale TTAGs is challenging. Each node might contain thousands of tokens, and the graph itself can have millions of time-stamped edges, with both text and structure evolving continuously. Jointly updating an LM and a GNN to handle this complexity often exceeds the memory and latency constraints of realistic applications [46, 17]. Staged approaches, such as first fine-tuning a language model and then using its embeddings as static node features for a GNN [5], can reduce computational overhead. However, they risk losing critical temporal signals, since static embeddings fail to capture evolving textual semantics and interaction patterns. This limitation is further amplified in *two-sided marketplaces*, where matching quality depends on continuously balancing interests from distinct user groups.

Our contributions. We introduce **GraphMatch**, a framework designed to address these challenges. GraphMatch integrates three key design elements:

1. **Scalable fusion of LMs and GNNs.** We begin by fine-tuning a Language Model (TextMatch) to derive domain-specific sentence embeddings. Then, GraphMatch employs these sentence embeddings using a lightweight residual projector. This strategy maintains a constant memory overhead per layer, enabling the GNN to iteratively refine representations by incorporating structural and temporal cues with rich textual semantics, thus enhancing scalability.

2. **Adversarial negative sampling.** We propose a novel approach to fine-tuning GNN with negative samples adversarial toward the LM embedding model. More specifically, after training TextMatch, we generate a training dataset for GraphMatch, which comprises recommendations generated by TextMatch with medium to high confidence. This sharpens the learning signal for GraphMatch, facilitating faster convergence toward improving the representations learned by TextMatch.

3. **A practical recipe for TTAGs for two-sided user recommendation.** Drawing on effective strategies for large textual graphs, our training approach combines (i) temporally matched positive pairs from observed interactions, (ii) *adversarially* mined negatives using TextMatch, and (iii) large in-batch negatives for efficient learning. We adapt our recipe for a two-sided marketplace recommendation system using task-homogeneous learning on mutually interesting objectives for both sides and effective handling of *cold-start* nodes. This comprehensive approach promotes stable convergence on graphs with tens of millions of nodes.

In this work, we present an extensive evaluation on a large-scale dataset derived user interaction data from Upwork [37], a leading work marketplace platform, demonstrating that GraphMatch consistently outperforms strong LM-only and GNN-only baselines, as well as LM-and-GNN fusion without temporal aspect. By offering a conceptually clear yet empirically powerful solution, GraphMatch advances representation learning on temporal, text-rich graphs in two sided marketplaces and provides a practical blueprint for future systems that need to reason about both language and evolving structure.

2 Related Work

Our research builds upon and extends several key areas within graph-based machine learning and recommendation systems. These include the application of Graph Neural Networks (GNNs) to recommendation tasks, techniques for learning representations from temporal and text-attributed graphs, the use of contrastive learning for graph representation, and the specific challenges of recommendation in two-sided marketplaces.

Graph Neural Networks for Recommendation. Graph Neural Networks (GNNs) have become a cornerstone in modern recommender systems due to their inherent ability to model complex relationships and capture collaborative signals within user-item interaction graphs [11]. By iteratively passing and aggregating messages between nodes, GNNs can learn rich, structure-aware embeddings for users and items, effectively leveraging higher-order connectivity [10, 38]. This structural modeling, combined with the ability to learn from node attributes, is beneficial for addressing challenges such as data sparsity and the cold-start problem within a recommendation ecosystem [27]. Seminal works like GraphSAGE [10] and its variants have demonstrated the power of neighborhood sampling and aggregation for inductive representation learning on large graphs. Our work leverages these principles to learn representations within the context of a two-sided marketplace.

Learning on Temporal and Text-Attributed Graphs. Real-world graphs, especially in domains like online marketplaces, are often temporal, with evolving structures and node attributes, and text-attributed, where nodes carry rich semantic information. Learning effectively from such Temporal Text-Attributed Graphs (TTAGs) requires models that can jointly process structural, textual, and temporal dimensions [41, 44]. Common approaches to represent structure and text together involve using LMs to generate initial node features, which are then refined by a GNN that captures graph structure [5, 4]. Other methods explore deeper fusion mechanisms between LMs and GNNs to better synergize semantic understanding with structural reasoning [42, 46, 19]. In contrast, GraphMatch proposes a scalable method to fuse LM-derived embeddings with a GNN that processes temporally-aware subgraphs while respecting textual semantics, an area of active research [45, 44].

Contrastive Learning for Graph Representations. Self-supervised learning, particularly through contrastive methods, has emerged as a powerful paradigm for learning graph representations without explicit labels [43, 13, 18]. Graph Contrastive Learning (GCL) typically involves generating multiple views of a graph (or nodes) through augmentations [33]. The choice of augmentation strategies and the selection of negative samples are critical for the success of GCL [36, 26, 9]. Techniques such as hard negative mining [29, 14] and adaptive negative sampling [40] aim to improve the quality of negative samples and thus the learned representations. GraphMatch utilizes contrastive objective in which empirically observed interactions are used as positive pairs, while TextMatch is used to mine adversarial hard negatives; this supply of temporally relevant positives and challenging negatives equips the GNN with precise signals that sharpen its representation learning.

Recommendation in Two-Sided Marketplaces. Two-sided marketplaces, such as freelance platforms [37] or e-commerce sites, present unique recommendation challenges that go beyond standard user-item matching [15, 20]. These platforms must balance the preferences and constraints of two distinct groups of users, such as clients and freelancers. Effective matching in such marketplaces requires training methodologies that can capture the nuances of both sides [8]. GraphMatch tackles this challenge with a lightweight, three-part strategy: (i) *temporal subgraph sampling* to track evolving interaction patterns (ii) *task-homogeneous mini-batches* that isolate client-side and freelancer-side signals during training and (iii) a *contrastive loss centered on "mutual-interest" events*, such as hires or interviews, which aligns the two sides, while we use TextMatch for adversarial hard negatives to improve discrimination.

3 TextMatch

We first use TextMatch to compute initial domain-adapted node representation denoted as $\Phi_{\text{text\_emb}}$: $(V \times \mathbb{R}) \to \mathbb{R}^{d_{\text{TM}}}$, which returns an embedding for any node $v \in V$ at any timestamp $t \in \mathbb{R}$, using only the textual content associated with the node. TextMatch embeddings serve as the initial semantic

input for the heterogeneous graph processed by GraphMatch. Its core objective is to generate powerful vector representations for entities of any type.

3.1 Model Architecture

We employ a Sentence-BERT style [28] encoder-only architecture. The parameters are shared between the query and candidate encoder, yielding a Symmetric Dual Encoder (SDE) model. This configuration is a common choice for top-performing text representation models on retrieval benchmarks such as BEIR [34] or MTEB [23].

3.2 Two-Stage Domain Adaptation and Fine-Tuning

To ensure that TextMatch embeddings are both generalizable and highly relevant to our specific domain, we adopt a two-stage training procedure (Figure 1) based on established multi-stage recipes prevalent in state-of-the-art embedding models [25, 39].

Stage 1: Weakly Supervised Pre-fine-tuning. Initially, the model is adapted using naturally occurring textual pairs and *weak* relevance signals sourced from the platform. Examples include *profile-title* $\leftrightarrow$ *profile-body* associations and dense signals from user interactions within the marketplace, such as *click* or *save* events. Consistent with findings that such weak supervision, combined with large batch sizes and in-batch negatives, yields robust generic embeddings for retrieval [39], we optimize temperature-scaled InfoNCE loss. In this stage, we prioritize large batch sizes and do not employ explicit hard-negative mining, allowing the model to learn broad semantic relationships while adapting to domain-specific vocabulary.

Stage 2: Supervised Fine-Tuning. We refine the model's representations using stronger *explicit* events as the primary supervisory signal. We use signals that show strong mutual interest between both sides of the marketplace, such as *interviews* or *hires*. For each positive pair, we incorporate a nuanced set of negaftives: (i) hard negatives, identified as chronologically adjacent impressions that *have not* converted into a meaningful interaction, thus representing plausible yet rejected alternatives [29], and (ii) random negatives sampled from the entire corpus to maintain distributional awareness. The InfoNCE loss is utilized again at this stage.

Training Stability and Two-Sided Marketplace Optimization. To optimally train for a two-sided marketplace, we construct *task-homogeneous* mini-batches. Each batch contains examples exclusively from *either* the client side *or* the freelancer side of the marketplace. This strategy addresses challenges reported in contrastive learning where mixing heterogeneous tasks within a batch can degrade retrieval quality [12]; specialized batching, such as the formation of single-task mini-batches, has been shown to mitigate this and improve the overall quality for model [21].

Outcome. The TextMatch pipeline produces domain-adapted and semantically rich node embeddings. These embeddings serve as the initial features for GraphMatch. This allows GraphMatch to dedicate its capacity to modeling temporal and global graph structure instead of learning basic textual similarity, forming an essential step in the overall system. We present the quantitative improvements attributable to this approach in Section 5.

4 GraphMatch

Graph definition. Similarly to TextMatch, GraphMatch learns numerical embeddings of entities in a user interaction graph. Let

$$G = (V, E, R, \mathcal{T}, \mathcal{A}, \Phi_{\text{text\_emb}}, \Phi_{\text{feat}}, \Psi) \tag{1}$$

denote the user interaction graph where V is the set of all nodes, R is the set of possible interaction types, and $E \subseteq V \times V \times R$ is the set of all edges. We define $\mathcal{T}$ as the set of node types (for example, freelancers, job posts, clients). Each node $v \in V$ has a type $\tau(v) \in \mathcal{T}$. Each edge is associated with a timestamp with $\Psi : E \to \mathbb{R}$.

We define $\mathcal{A} : V \to \mathcal{P}(\mathbb{R} \times \mathbb{R})$ as the activity periods function that maps each node $v \in V$ to a set of activity periods, where we represent each activity period as a tuple $(t_{\text{start}}, t_{\text{end}})$ with $t_{\text{start}} \leq t_{\text{end}}$.

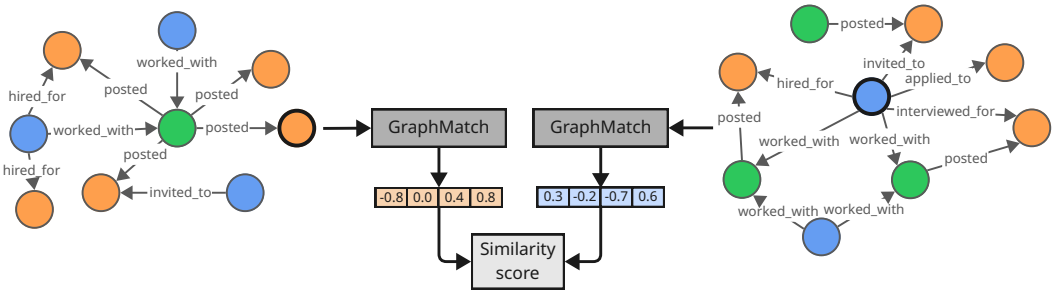

Figure 2: GraphMatch embeds freelancers, clients or job posts using a sampled text-attributed graph representing their work history. In the illustration, we predict the embeddings of the emboldened job post (left) and freelancer (right) nodes using the surrounding graph. We compare GraphMatch embedding vectors using cosine similarity to predict the match probability between two entities.

The set $\mathcal{A}(v)$ can be empty (indicating no recorded activity periods) or contain one or more non-overlapping time intervals during which node v was *active*. *Activity period* may pertain to when a freelancer was open for work or when a job post was requesting applications.

$\Phi_{\text{text\_emb}} : (V \times \mathbb{R}) \to \mathbb{R}^{d_{\text{TM}}}$ maps every node and timestamp to its TextMatch embedding vector. We assume that TextMatch embedding is set to a zero vector for nodes without textual content. We store temporal TextMatch embedding versions per node in a sorted table as depicted in Figure 3. For each node type $\tau \in \mathcal{T}$, we define a type-specific feature function $\Phi^{\tau}_{\text{feat}} : \{(v,t) \in V \times \mathbb{R} | \tau(v) = \tau\} \to \mathbb{R}^{d_\tau}$, where d_τ is the dimension of numerical features for nodes of type τ. GraphMatch leverages this heterogeneous graph structure to learn an embedding function $f : V \to \mathbb{R}^{d_{\text{GM}}}$ that captures both textual similarity and interaction patterns to enhance matching performance.

Subgraph sampling. In practice, using the full graph to compute node embeddings is computationally prohibitive. In Figure 2, we show how GraphMatch utilizes neighborhood subgraphs to compute entity embeddings. To construct the input graph for GraphMatch, given target node $v \in V$ and query timestamp $T \in \mathbb{R}$, we sample a temporal subgraph up to K hops away from the target node. At each hop, for each edge type $r \in R$, we retain up to N edges $e \in E$ such that $\Psi(e) \leq T$, selecting the N most recent such edges if more than N exist. Node features are also sampled as of timestamp T, using binary search over time-indexed feature snapshots to retrieve the most recent features prior to T (see Figure 3). This temporal subgraph construction enables GraphMatch to reason over temporally relevant interactions while maintaining efficiency.

4.1 Model Architecture

In this section, we describe the GraphMatch model. The GraphMatch framework is agnostic to some modeling choices such as the implementation of graph convolutional layer. We describe the key elements of our approach.

Initial Node Representation. The initial representation of a node $v \in V$ depends on its type $\tau(v) \in \mathcal{N}$ and timestamp $t \in \mathbb{R}$. For each node type $n \in \mathcal{N}$, we have a type-specific encoder network Encoder_n that processes the node features:

$$h^{(0)}_{v,t} = \text{Encoder}_{\tau(v)}\Big[\Phi_{\text{text\_emb}}(v,t) \,\|\, \Phi^{\tau(v)}_{\text{feat}}(v,t)\Big] \tag{2}$$

Each encoder network Encoder_n is a small feed-forward neural network that maps the node type-specific input features to a common embedding space.

Graph Convolution. GraphMatch applies multiple layers of heterogeneous graph convolution to learn interaction-aware representations of nodes. Each heterogeneous graph convolution layer calculates

$$h^{(l+1,r)}_{v,t} = \text{CONV}_r\left(h^{(l)}_{v,t}, \{h^{(l)}_{u,t} \mid u \in \mathcal{N}_r(v,t,K)\}\right),$$

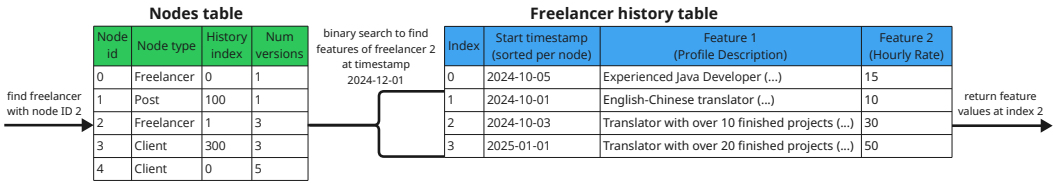

Figure 3: We store dynamic node features in two tables. The main table (left) contains a single row per node, with its history start index and the number of versions. The node history table (right) stores all available versions of features per each node, grouped by node and sorted by timestamp within each node. Given any timestamp, node type, and node ID, we first query the main table to retrieve the history index and the number of versions. Next, we run a binary search over the relevant rows in the feature history table, finding the point-in-time correct feature values in $O(log_2 n)$ time.

where $h_{v,t}^{(l)}$ denotes the representation of node v at layer l, $\mathcal{N}_r(v, t, K)$ is the set of the K most recent neighbors of v connected by relation type r with edge timestamp $< t$, and CONV_r is a relation-specific aggregation function. The representations from each relation are aggregated (for example, by averaging) to obtain $h_{v,t}^{(l+1)}$. This process is repeated for multiple layers to capture higher-order interactions.

Final Embedding Fusion. After the final graph convolution layer, the model applies a residual connection to the original TextMatch embedding:

$$f(v,t) = \frac{h_{v,t}^{\text{final}} + \text{Project}(\Phi_{\text{text\_emb}}(v,t))}{\left\| h_{v,t}^{\text{final}} + \text{Project}(\Phi_{\text{text\_emb}}(v,t)) \right\|_2}, \tag{3}$$

where Project is a small feed-forward module that maps the TextMatch embedding to the final embedding space. This design ensures that the final embedding can benefit from TextMatch embeddings, numerical node features, and the graph-based learned representations.

4.2 Training Procedure

We train GraphMatch using a contrastive learning framework. The goal is to learn embeddings that place relevant query-candidate pairs closer in the embedding space while pushing away irrelevant pairs.

Positive Training Labels. We train GraphMatch for a temporal link prediction task. The set of training labels

$$C = \{(q_i, v_i^+, T_{\text{start}}, T_{\text{end}})\} \subseteq V \times V \times \mathbb{R} \times \mathbb{R} \tag{4}$$

is derived from observed matches, where each match is associated with a relevance time span $(T_{\text{start}}, T_{\text{end}})$. For example, if a match represents a contract between a job post and a freelancer, the associated T_{start} is the timestamp when the job offer was created, and T_{end} is the timestamp when the contract was started. Note that this time span does not have to be equal to an activity time span from $\mathcal{A}$. For example, some job posts may request many freelancers and remain open for applications after a contract is initiated. For each sample during training, we randomly select a timestamp t_i^+ uniformly between T_{start} and T_{end} and make a prediction based on the version of the graph at t_i^+.

Adversarial Negative Mining. We want GraphMatch to improve upon TextMatch embeddings by taking advantage of the user interaction graph. To achieve this, we employ negative samples generated *adversarially* toward TextMatch. For each positive sample (q_i, v_i^+, t_i^+), we generate up to N adversarial negative samples by first building separate Approximate Nearest Neighbor search [31] indices for each node type using their respective activity periods, then performing similarity-based filtering to select challenging negatives. We present the algorithm pseudocode in Algorithm 1. In our experiments, we use $\sigma_{\text{low}} = 0.5$, $\sigma_{\text{high}} = 0.85$, $K = 2000$, and $N = 20$.

Additionally, per each positive sample (q_i, v_i^+, t_i^+), we generate a random negative candidate (v_i^-, t_i^-) such that $\tau(v_i^-) = \tau(v_i^+)$ and $\exists (t_{\text{start}}, t_{\text{end}}) \in \mathcal{A}(v_i^-) : t_{\text{start}} \leq t_i^- \leq t_{\text{end}}$. This ensures that the space

Algorithm 1 Adversarial Negative Mining via TextMatch.

Require: Positive dataset $(q_i, v_i^+, t_i^+) \in C \subseteq V \times V \times \mathbb{R}$
Require: TextMatch embedding function $\Phi_{\text{text\_emb}} : (V \times \mathbb{R}) \to \mathbb{R}^{d_{TM}}$
Require: Activity periods function $\mathcal{A} : V \to \mathcal{P}(\mathbb{R} \times \mathbb{R})$
Require: Node type function $\tau : V \to \mathcal{T}$
Require: Parameters: K, N, σ_{low}, σ_{high}
Build ANN indices $\mathcal{I}_{\mathcal{T}} \leftarrow \{\mathcal{I}_t : t \in \mathcal{T}\}$ where each $\mathcal{I}_t$ contains embeddings of nodes v with $\tau(v) = t$ and active periods in $\mathcal{A}(v)$
$\mathcal{D}_{\text{train}} \leftarrow \emptyset$
for each $(q_i, v_i^+, t_i^+) \in C$ **do**
    $e_{q_i} \leftarrow \Phi_{\text{text\_emb}}(q_i, t_i^+)$
    $\mathcal{C}_K \leftarrow \text{ANN}(e_{q_i}, K, \mathcal{I}_{\tau(v_i^+)})$ ▷ Nearest K within same node type
    $\mathcal{F} \leftarrow \emptyset$
    for each $(v_j^-, t_{\text{start}}^-, t_{\text{end}}^-) \in \mathcal{C}_K$ **do**
        $t_j^- \leftarrow \text{RandomUniform}(t_{\text{start}}^-, t_{\text{end}}^-)$
        $e_{v_j^-} \leftarrow \Phi_{\text{text\_emb}}(v_j^-, t_j^-)$
        $s_j^- \leftarrow \text{sim}(e_{q_i}, e_{v_j^-})$
        if $\sigma_{\text{low}} < s_j^- < \sigma_{\text{high}}$ and $v_j^- \neq v_i^+$ **then**
            $\mathcal{F} \leftarrow \mathcal{F} \cup \{(v_j^-, t_j^-)\}$
        end if
    end for
    $\mathcal{N}_i \leftarrow \text{RandomSample}(\mathcal{F}, \min(N, |\mathcal{F}|))$
    for each $(v_j^-, t_j^-) \in \mathcal{N}_i$ **do**
        $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \{(q_i, v_i^+, t_i^+, v_j^-, t_j^-)\}$
    end for
end for
return $\mathcal{D}_{\text{train}}$

of negative samples covers all potential active nodes. In Section 5, we show that the combination of random and adversarial negatives is required to surpass the performance of TextMatch.

In-Batch Negative Sampling. To enhance computational efficiency and exposure to diverse negative examples, we leverage in-batch negatives, similar to TextMatch. Assume our batch consists of queries $\{(q_i, t_i^+)\}$, positive candidates $\{(v_i^+, t_i^+)\}$ per each query and the set of all random and adversarial negative candidates generated for the batch $\{(v_j^-, t_j^-)\}$. We compute the similarities between each query q_i and all positive and negative candidate entities in the batch:

$$s_{i,j}^+ = f(q_i, t_i^+)^T f(v_j^+, t_j^+) \tag{5}$$

$$s_{i,j}^- = f(q_i, t_i^+)^T f(v_j^-, t_j^-) \tag{6}$$

where $f(\cdot)$ denotes the GraphMatch embedding function.

Contrastive Loss. We formulate the training objective as a cross-entropy loss where, for each query, the model must identify its corresponding positive candidate among all candidates in the batch:

$$\mathcal{L} = -\frac{1}{B} \sum_{i=1}^{B} \log \frac{\exp(s_{i,i}^+/\tau)}{\sum_{j \in C^+} \exp(s_{i,j}^+/\tau) + \sum_{j \in C^-} \exp(s_{i,j}^-/\tau)} \tag{7}$$

where B is the batch size, i^+ is the index of the positive candidate for query i, C^+ and C^-, respectively, represent the sets of all positive and negative candidate indices in the batch, and $\tau > 0$ is the temperature hyperparameter.

Model	Adv Neg	NDCG@10	
		FL → JP	JP → FL
snowflake-arctic-embed-l		7.4 %	6.1 %
mxbai-embed-large-v1		11.2 %	6.7 %
TextMatch-small		21.6 %	10.6 %
TextMatch-large		22.4 %	11.4 %
GraphMatch-no-text	×	7.4 %	3.3 %
	✓	12.8 %	7.4 %
GraphMatch-no-feat	×	21.1 %	10.4 %
	✓	22.4 %	11.8 %
GraphMatch-full	×	21.4 %	10.3 %
	✓	**24.2** %	**12.4** %

Table 1: Performance comparison on FL → JP and JP → FL retrieval, including GraphMatch trained with or without adversarial negatives.

5 Experiments

We train TextMatch and GraphMatch on two-sided work marketplace data. The training objective is to predict *contracts* between freelancers and job posts based on freelancer profile overview and job description, and user interaction graph in the case of GraphMatch. We train TextMatch and GraphMatch in a multitask fashion, using training labels for predicting job posts for a query freelancer (FL → JP) or a freelancer for a query job post (JP → FL) with 50%/50% sampling probabilities. We depict the full training flowchart in Figure 1.

5.1 TextMatch Training

The encoder is initialized from publicly available `E5-unsupervised` checkpoints (*small*: 33M and *large*: 330M parameters), which are themselves results of advanced contrastive pretraining [39]. We use the same encoder weights for queries and candidates. The pre-fine-tuning stage utilizes dense interaction data of 110M samples based on *weak* signals such as clicks or saves and naturally occurring pairs of titles and descriptions. The fine-tuning stage uses more sparse but *strong* interactions, which show mutual interest between both sides of the marketplace, like interviews or a hires, combined with hard negatives based on the impressed but not interacted items mined from interaction logs.

5.2 GraphMatch Training

Training dataset. The dataset for GraphMatch contains one year of user interaction data from Upwork [37], with a 10/1/1 month temporal split for training, validation, and evaluation. The dataset consists of nodes representing freelancers, clients and job posts, and edges representing interactions such as posting a job by a client, job application, freelancer invitation, job interview, or contract start (see Figure 2). Freelancer and job post nodes are associated with textual features: profile overview and job description, respectively. All nodes contain additional numerical features specific to their type, such as freelancer hourly rate, client location, job post category, etc. Both text and numerical features can change over time. In total, there are approximately 9 million nodes with 32 million temporal versions of features and 62 million edges. We train GraphMatch on *strong* labels derived from contracts between freelancers and job posts.

Training details. We train GraphMatch using TextMatch-small embeddings for Φ_{text_emb}. The output embedding dimension for GraphMatch is $d_{GM} = 1024$. We freeze the weights of TextMatch and only update the weights of the GraphMatch-specific layers. We train GraphMatch for 24 hours, with hourly checkpointing, and for the final evaluation, select the checkpoint with the lowest validation loss. We use GATv2Conv [3] as the graph convolutional layer. In total, our model has approximately 64M weights, including 33M weights of TextMatch-small. We use a cluster of 8xNVIDIA A100 80GB GPUs for all training and evaluation.

Model	NDCG@10	
	FL → JP	JP → FL
GraphMatch-full	**24.2** %	**12.4** %
GraphMatch-full (no temporal nodes)	15.1 %	8.5 %
GraphMatch-full (no temporal graph)	13.3 %	6.9 %

Table 2: Ablation study on disabling temporal sampling during training.

Model	NDCG@10 (FL → JP)		
	Query CS	Cand CS	Both CS
TextMatch-small	21.6 %	23.4 %	**23.3** %
TextMatch-large	21.7 %	**24.1** %	22.7 %
GraphMatch-no-text	5.7 %	6.9 %	5.8 %
GraphMatch-no-feat	24.1 %	23.2 %	21.9 %
GraphMatch-full	**25.3** %	23.0 %	22.8 %

Table 3: Evaluation on samples where query, ground truth candidate or both of these entities are *cold start* (CS) nodes.

5.3 Evaluation

Evaluation data. The evaluation set comprises contracts between freelancers and job posts from 14 consecutive days within the evaluation part of the graph. For each evaluation day, as the set of candidates, we gather all active freelancers and all job posts open for hire as of 12:00 GMT that day. Out of these, we select as positive pairs the pairs of freelancers and job posts that engaged in a mutual contract with the contract start date after 12:00 GMT on that day. We evaluate for two mirrored retrieval tasks: retrieving the correct job post based on a query freelancer (FL → JP), and retrieving the correct freelancer based on a job post (JP → FL). For the JP → FL task, we only consider candidate freelancers who have completed their profile (profile completion > 80%) and were active on the website within 48 hours prior to the considered timestamp. The dataset contains ∼15000 positive labels, with ∼13000 job posts per day and ∼24000 freelancers per day as the candidate pool.

Retrieval metrics. In Table 1, we show the aggregated NDCG@10 metric values for retrieval tasks FL → JP and JP → FL on the evaluation set. We compare TextMatch and GraphMatch with two state-of-the-art open-source embeddings models: *snowflake arctic-embed-l* [22] and *mxbai-embed-large-v1* [16]. We use the open-source model weights without any fine-tuning in a zero-shot fashion. We see that fine-tuning on the work marketplace dataset improves the accuracy of the text embedding model, exemplified by the high NDCG@10 of TextMatch compared to the open-source embedding models. The performance gap is significant despite the large size and generalized pre-training of the open source baselines, highlighting the importance of fine-tuning on the target dataset.

For GraphMatch, we observe that the combination of random and hard negatives is crucial for good performance. GraphMatch versions trained only with random negatives achieve performance comparable to TextMatch-small. We ablate over node features used by GraphMatch: *no-feat* uses only TextMatch node embeddings, *no-text* uses numerical node features without text embeddings, and *full* uses the concatenation of TextMatch node embeddings and numerical node features. GraphMatch-full, trained with adversarial negatives, achieves the best performance, while GraphMatch-no-feat performs comparably to TextMatch-large, despite having a much smaller model size. We observe that textual context is crucial for accurate predictions, showcased by the low accuracy of GraphMatch-no-text.

Importance of temporally accurate sampling. Sampling point-in-time correct subgraphs and node features is necessary to train a model that generalizes well to unseen nodes. To verify this, we train GraphMatch in two setups. The *no-temporal-nodes* version does not use temporally relevant node features during training (Figure 3) but always utilizes the newest feature version in the training set. The *no-temporal-graph* version additionally does not sample temporally relevant subgraphs but has access to all the graph edges during training. In Table 2, we show that such sampling leads to poor generalization to the evaluation set, with much lower NDCG@10 values.

Retrieval accuracy on cold start nodes. The *cold start problem* is a critical challenge for recommender systems. In our setting, a cold start node can be a freelancer who has not yet interacted with other users or job posts, or a job post posted by a client who has not had any other interactions so far.We evaluate the performance of TextMatch and GraphMatch (trained with adversarial negatives) on the evaluation samples where the query or ground truth candidate entity is a cold start node. Table 3 presents the evaluation results on samples for which query or ground truth candidate is a

cold start node. We see that the performance of GraphMatch on cold start nodes is comparable to TextMatch, indicating the model's ability to default to using text embeddings where no graph information is available, mitigating the cold start problem.

6 Real-time Deployment of GraphMatch

The real-time deployment architecture of GraphMatch consists of three main components: *feature store*, *inference service*, and *graph database*. This separation supports a shared, universal graph structure across various use cases while allowing individual GNN models to leverage customized node features. We implement node sampling using Cypher [7], letting callers define the node and edge types needed at inference.

Feature Store. Node features are primarily aggregated statistics (counts, averages, etc.). Our data warehouse, Snowflake [30], stores clones of production relational databases and assorted business data. Feature tables are generated via hourly SQL Mesh [32] ETL jobs and maintained in Feast [6]. Since features update hourly, we define smart default values to handle missing features for entities present in the graph but not yet in the feature store.

Graph Database. We host graph using Neo4j Aura [24] graph database. Two ETL pipelines update the graph: a near real-time Kafka [2] pipeline for highly dynamic entities like job posts and an hourly Airflow [1] batch pipeline for other entities. In the graph database, we store minimal node properties (type, ID, creation timestamp) to enable accurate subgraph sampling for inference.

Inference Service. Model inference runs on a Python/FastAPI [35] microservice. We deploy the model on EC2 instances with Nvidia A10G GPUs (24GB RAM). Requests per second drive autoscaling, as GPU/CPU utilization is less predictive of latency degradation as usage increases. To optimize model inference, we compute and cache TextMatch embeddings for nodes every time a freelancer profile or job post description is updated, effectively limiting online inference to only the GNN layers. This enables achieving average embedding generation latencies of less than 70 ms per embedding across all entity types, supporting horizontal scalability and unlocking near real-time use cases.

7 Conclusion

In this work, we introduced **GraphMatch**, a framework for learning entity representations in a two-sided marketplace by integrating textual attributes and evolving user-interaction patterns using Temporal Text Attributed Graphs (TTAGs). Our experiments on a large-scale dataset from a real-world labor marketplace demonstrate that GraphMatch significantly outperforms text-only embedding approaches for matching freelancers with job posts. We show that effectively fusing textual and graph embeddings, with appropriate temporal sampling, achieves optimal performance. Additionally, we provided practical guidelines for deploying GraphMatch as a real-time, graph-based recommendation system.

Although GraphMatch is effective in recommendation tasks, it has few limitations. Its deployment complexity exceeds that of text-only embeddings, primarily because it requires real-time access to TTAGs during inference. Furthermore, the multi-stage training process of TextMatch and GraphMatch, while computationally expensive, is essential for accurately capturing textual, temporal, and structural features inherent to TTAGs.

Several directions for future research could enhance our approach. Firstly, incorporating a more sophisticated fusion mechanism, as suggested in [46, 42], may further improve the integration of textual and graph embeddings. Secondly, exploring multi-stage training for GraphMatch akin to TextMatch could enhance embedding quality. Lastly, deeper investigations into node and edge type definitions or subgraph sampling strategies is interesting. Ultimately, although we focused on work marketplace, the principles and methodologies underpinning GraphMatch are broadly applicable to any two-sided marketplace.

References

[1] Apache Org. Airflow. platform created by the community to programmatically author, schedule and monitor workflows. `https://airflow.apache.org/`. Accessed: 2025-05-23.

[2] Apache Org. Kafka. open-source distributed event streaming platform. `https://kafka.apache.org/`. Accessed: 2025-05-23.

[3] S. Brody, U. Alon, and E. Yahav. How attentive are graph attention networks? *arXiv preprint arXiv:2105.14491*, 2021.

[4] E. Chien, W.-C. Chang, C.-J. Hsieh, H.-F. Yu, J. Zhang, O. Milenkovic, and I. S. Dhillon. Node feature extraction by self-supervised multi-scale neighborhood prediction. In *International Conference on Learning Representations (ICLR)*, 2022.

[5] K. Duan, Q. Liu, T.-S. Chua, S. Yan, W. T. Ooi, Q. Xie, and J. He. Simteg: A frustratingly simple approach improves textual graph learning, 2023.

[6] Feast. Serving data for production ai. `https://feast.dev`. Accessed: 2025-05-23.

[7] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 1433–1445, New York, NY, USA, 2018. Association for Computing Machinery.

[8] A. Goswami, F. Hedayati, and P. Mohapatra. Recommendation systems for markets with two sided preferences. In *2014 13th International Conference on Machine Learning and Applications*, pages 282–287, 2014.

[9] H. Hafidi, M. Ghogho, P. Ciblat, and A. Swami. Negative sampling strategies for contrastive self-supervised learning of graph representations. *Signal Processing*, 190:108310, 2022.

[10] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[11] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.

[12] S. Hofstätter, S.-C. Lin, J.-H. Yang, J. Lin, and A. Hanbury. Efficiently teaching an effective dense retriever with balanced topic aware sampling. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 113–122, 2021.

[13] W. Ju, Y. Gu, X. Luo, Y. Wang, H. Yuan, H. Zhong, and M. Zhang. Unsupervised graph-level representation learning with hierarchical contrasts. *Neural Networks*, 158:359–368, 2023.

[14] Y. Kalantidis, M. B. Sariyildiz, N. Pion, P. Weinzaepfel, and D. Larlus. Hard negative mixing for contrastive learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[15] J. Katukuri, R. Mukherjee, T. Konik, and T. Konik. Large-scale recommendations in a dynamic marketplace. In *Workshop on large scale recommendation systems at RecSys*, volume 13, 2013.

[16] S. Lee, A. Shakir, D. Koenig, and J. Lipp. Open source strikes bread - new fluffy embedding model. https://www.mixedbread.ai/blog/mxbai-embed-large-v1, 2024.

[17] A. Longa, V. Lachi, G. Santin, M. Bianchini, B. Lepri, P. Lio, F. Scarselli, and A. Passerini. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities, 2023.

[18] X. Luo, W. Ju, Y. Gu, Z. Mao, L. Liu, Y. Yuan, and M. Zhang. Self-supervised graph-level representation learning with adversarial contrastive learning. *ACM Trans. Knowl. Discov. Data*, 18(2), Nov. 2023.

[19] C. Mavromatis, V. N. Ioannidis, S. Wang, D. Zheng, S. Adeshina, J. Ma, H. Zhao, C. Faloutsos, and G. Karypis. Train your own gnn teacher: Graph-aware distillation on textual graphs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 157–173. Springer, 2023.

[20] R. Mehrotra, J. McInerney, H. Bouchard, M. Lalmas, and F. Diaz. Towards a fair marketplace: Counterfactual evaluation of the trade-off between relevance, fairness & satisfaction in recommendation systems. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 2243–2251, New York, NY, USA, 2018. Association for Computing Machinery.

[21] R. Meng, Y. Liu, S. R. Joty, C. Xiong, Y. Zhou, and S. Yavuz. Sfr-embedding-mistral:enhance text retrieval with transfer learning. Salesforce AI Research Blog, 2024. Accessed: 2025-05-23.

[22] L. Merrick. Embedding and clustering your data can improve contrastive pretraining. https://arxiv.org/abs/2407.18887, 2024.

[23] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers. Mteb: Massive text embedding benchmark, 2022.

[24] Neo4j. Neo4j auradb: Fully managed graph database. `https://neo4j.com/product/auradb/`. Accessed: 2025-05-23.

[25] J. Ni, C. Qu, J. Lu, Z. Dai, G. H. Ábrego, J. Ma, V. Y. Zhao, Y. Luan, K. B. Hall, M.-W. Chang, and Y. Yang. Large dual encoders are generalizable retrievers, 2021.

[26] K. Nozawa and I. Sato. Understanding negative samples in instance discriminative self-supervised representation learning. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, Red Hook, NY, USA, 2021. Curran Associates Inc.

[27] T. Qian, Y. Liang, Q. Li, and H. Xiong. Attribute graph neural networks for strict cold start recommendation. *IEEE Transactions on Knowledge and Data Engineering*, 34(8):3597–3610, Aug. 2022.

[28] N. Reimers and I. Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In K. Inui, J. Jiang, V. Ng, and X. Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, Nov. 2019. Association for Computational Linguistics.

[29] J. Robinson, C.-Y. Chuang, S. Sra, and S. Jegelka. Contrastive learning with hard negative samples. *arXiv preprint arXiv:2010.04592*, 2020.

[30] Snowflake Inc. Snowflake: Cloud data platform. `https://www.snowflake.com/`. Accessed: 2025-05-23.

[31] Spotify. Annoy library. `https://github.com/spotify/annoy`. Accessed: 2025-05-23.

[32] SQL Mesh. Data transformation built for growth. `https://sqlmesh.com/`. Accessed: 2025-05-23.

[33] S. Thakoor, C. Tallec, M. G. Azar, M. Azabou, E. L. Dyer, R. Munos, P. Veličković, and M. Valko. Large-scale representation learning on graphs via bootstrapping. *International Conference on Learning Representations (ICLR)*, 2022.

[34] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[35] Tiangolo. Fastapi framework: high performance, easy to learn, fast to code, ready for production. `https://fastapi.tiangolo.com/`. Accessed: 2025-05-23.

[36] P. Trivedi, E. S. Lubana, Y. Yan, Y. Yang, and D. Koutra. Augmentations in graph contrastive learning: Current methodological flaws towards better practices, 2022.

[37] Upwork. The world's work marketplace. https://www.upwork.com/.

[38] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[39] L. Wang, N. Yang, X. Huang, B. Jiao, L. Yang, D. Jiang, R. Majumder, and F. Wei. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*, 2022.

[40] Y. Wang, L. Hu, W. Gao, X. Cao, and Y. Chang. Adans: Adaptive negative sampling for unsupervised graph representation learning. *Pattern Recognition*, 136:109266, 2023.

[41] H. Yan, C. Li, R. Long, C. Yan, J. Zhao, W. Zhuang, J. Yin, P. Zhang, W. Han, H. Sun, et al. A comprehensive study on text-attributed graphs: Benchmarking and rethinking. *Advances in Neural Information Processing Systems*, 36:17238–17264, 2023.

[42] J. Yang, Z. Liu, S. Xiao, C. Li, D. Lian, S. Agrawal, A. Singh, G. Sun, and X. Xie. Graphformers: Gnn-nested transformers for representation learning on textual graph. *Advances in Neural Information Processing Systems*, 34:28798–28810, 2021.

[43] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen. Graph contrastive learning with augmentations. *Advances in neural information processing systems*, 33:5812–5823, 2020.

[44] J. Zhang, J. Chen, M. Yang, A. Feng, S. Liang, J. Shao, and R. Ying. Dtgb: A comprehensive benchmark for dynamic text-attributed graphs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 91405–91429. Curran Associates, Inc., 2024.

[45] S. Zhang, Y. Xiong, Y. Tang, X. Chen, Z. Jia, Z. Gu, J. Xu, and J. Zhang. Unifying text semantics and graph structures for temporal text-attributed graphs with large language models, 2025.

[46] J. Zhao, M. Qu, C. Li, H. Yan, Q. Liu, R. Li, X. Xie, and J. Tang. Learning on large-scale text-attributed graphs via variational inference. *International Conference on Learning Representations*, 2023.