Journal of Applied Automation Technologies

Research Paper

Application of Large-Scale Subgraph Matching Algorithms in Natural Language Processing

Edmund Kapuściński[1] and Marcel Barczyk[1, *]

[1] Faculty of Computer Science, AGH University of Science and Technology, Krakow, 30-059, Poland
*Corresponding author: marcel.ba@agh.edu.pl

Abstract. Large-scale natural language processing increasingly needs to have structured evidence that can link entities, predicates, events and discourse relations in massive corpora. This paper investigates the application of large-scale subgraph matching algorithms in NLP and proposes LSM-NLP, a hybrid framework that combines typed textual graph construction, structural indexing, candidate verification and neural re-ranking. The framework regards subgraph matching as an engineering problem of symbolic linguistic structure and neural semantic representation. It has been shown through experiments that it can perform relation extraction, semantic retrieval, question-answer evidence selection and document clustering. According to the above planned results, LSM-NLP will increase the average F1 score to 80.7 from 75.4 and lower the median latency to 612ms at the 10-million-node graph scale, keeping shard-level score drift below 0.033 after calibration. Based on the above studies, a scalable subgraph matching can be used to improve both the interpretability and efficiency of precise relational evidence in NLP systems.

Keywords: *Large-scale Subgraph Matching, Natural Language Processing, Typed Textual Graph, Neural Re-ranking, Evidence Retrieval*

Received on 19 June 2022, Accepted on 28 January 2023, Published on 19 February 2023

Introduction

Natural Language Processing is now a large-scale engineering discipline concerned with organising documents, conversations, entities, claims and events as structured evidence rather than as individual token sequences [1]. Transformer encoders have enhanced contextual representation, but their dense attention maps do not directly show the syntactic and semantic relationships required by many downstream decisions [2]. Therefore, knowledge graphs and text-derived graphs have been used to connect local linguistic cues with external relational knowledge [3]. Dependency arcs, semantic roles, entity links, discourse relations and co-reference chains are all patterns that are difficult to retain in a purely sequential model [4]. Modern retrieval and extraction systems also need to find repeated relational patterns in millions of passages, and the same meaning may be expressed in different words [5]. Graph neural networks provide one way to learn from such structures, but they generally assume that the relevant subgraph has already been identified [6]. Classical graph matching is another way that explicitly searches for structural patterns, but it needs to be redesigned for noisy and large text graphs [7]. The intersection of the two traditions motivates a scalable subgraph matching layer for interpretable NLP [8].

The main problem is that subgraph matching is computationally expensive when the data graph has millions of nodes and various kinds of relations [9]. Exact subgraph isomorphism can offer strong structural guarantees, but it is not feasible to enumerate all candidate node mappings [10]. Approximate graph search reduces computation but may eliminate some valid paraphrastic or parser-noisy matches through aggressive pruning [11]. Neural semantic retrieval can handle lexical variation, but it often retrieves passages that are semantically related but do not meet the required relation structure [12]. Therefore, the NLP system needs to be applied in

practice and, by means of structure and semantics, reduce the search space and address linguistic variations [13]. The way should also provide evidence for inspection by engineers and domain users to resolve opaque retrieval failures in a high-stakes environment [14]. No one has yet created a full engineering account of how large-scale subgraph matching should be formulated, indexed, scored and evaluated for NLP workloads in the existing studies [15].

Given the above deficiency, this paper puts forward a large-scale subgraph matching framework for natural language processing. Each document is viewed as a kind of text graph in the framework, task requirements are reduced to short query graphs, candidate regions are acquired via structural signatures, and finally, after verification, these candidates are re-ranked by a light-weight neural scoring module. It is not a new parser or a new language model; it is an algorithm that links graph structure and neural semantics under the constraint of a real corpus scale. Provide a concrete experimental procedure in the paper, include data values, diagnostic indicators, and chart placeholders, and thus be applicable to relation extraction, semantic retrieval, evidence selection and document clustering.

Related Work

Graph-Based Representation in NLP

Graph-based NLP starts with the idea that language has relationships and, although token order can provide some of this information, it is insufficient [16]. Dependency parsing and semantic role labelling reveal the predicate-argument structure and are used to identify who performed which action on whom by a system [17]. Entity linking and knowledge graph construction connect local mentions with normalised concepts to expand the relational context for document interpretation [18]. Graph neural networks are extensions of this idea that learn representations by performing message passing over nodes and edges, rather than using only sentence-level vectors [19]. However, graph neural networks generally perform better when the relevant local structure is already available, and thus candidate subgraph discovery is a prerequisite issue [20].

Subgraph Matching and Scalable Search

Subgraph matching studies whether a query graph can be mapped to a larger graph that retains node labels, edge labels and adjacency constraints [21]. Large-graph search systems reduce the candidate space using signatures, path indexes, neighbourhood filters and partitioned execution [22]. Approximate subgraph matching relaxes the rigid structure requirements and is relatively convenient for natural language processing (NLP) because parser errors, paraphrasing, ellipsis, and discourse variation all alter graph topology [23]. Neural retrieval methods can supplement graph search by assigning a score to the semantic similarity of implausible candidates after structural filtering [24]. The last gap in research is how to combine the above components into an all-encompassing NLP architecture that is scalable, interpretable and measurable at the task level [25].

Proposed Methodology

Typed Textual Graph Representation

First, create typed textual graphs of the corpus in the proposed method. Each document graph contains token nodes, entity mention nodes, normalised entity nodes, predicate nodes, event nodes, sentence nodes and optional discourse nodes. Edges are dependency relations, semantic roles, entity normalisation links, co-reference links, sentence adjacency, discourse relations and task-specific retrieval links. This heterogeneous construction is deliberately conservative; it does not require all relations to be of the same edge type, as a dependency edge and an entity normalisation edge have different purposes in matching.

A Query Graph is a local structure required by the task. A query in relation extraction may contain two entity nodes connected by a predicate and a semantic role path. In question answering, the query may contain an answer type, an entity constraint and an evidence sentence linked by discourse or co-reference edges. In document clustering, the query may be a frequent motif that represents a repeated technical claim. The query graph is small, typed and weighted; the document graph is large, noisy and distributed across shards.

$$G_d = (V_d, E_d, T_V, T_E, X_d) \quad \text{Eq.(1)}$$

Document graph G_d contains nodes, edges, node types, edge types, and contextual node features.

Separately store the symbol label and context embedding in the representation. Symbolic labels are employed for indexing and hard pruning, and embeddings are used later for semantic re-ranking. Therefore, it can avoid the cost of a neural comparison when a candidate has already violated the basic type or edge constraints. It is also easier to design the inspection; thus, a failure can be attributed to a missing structure, weak semantic similarity, or task-level mismatch.

$$Q = \left(V_q, E_q, L_q, W_q\right), \left|V_q\right| \ll |V_d| \quad \text{Eq.(2)}$$

Query graph Q is intentionally much smaller than the document graph and stores labels and task weights.

The graph builder is deterministic; thus, repeated indexing results in the same node identifiers. First, segment and parse the sentences; normalize entity mentions; detect predicate candidates; and only after forming a local syntactic structure add sentence-level evidence links. The graph builder saves confidence values for dependency arcs, entity links and discourse edges. Low-confidence edges are not deleted immediately; they may still provide some helpful soft evidence for matching. They are therefore not included in the main group and will be considered later under the penalty term of the structural feasibility score.

$$s(v) = hash(T_V(v), label(v), deg(v), M_E(v)) \quad \text{Eq.(3)}$$

Node signature $s(v)$ summarizes type, label, degree, and incident edge-type multiset.

This representation also supports domain adaptation. Entity types in a biomedical corpus may include gene, disease, chemical, treatment, and phenotype; those in a legal corpus can be statute, obligation, actor, condition, exception and date. The matching algorithm does not need to be rewritten for each domain because node types and edge types are read from the schema. A new domain only needs schema extension, graph construction rules and task templates. It needs to be an engineering deployment that offers reusable infrastructure for subgraph matching, not a single-use extraction script.

The Graph Schema also needs to retain the origin information. Each node stores the sentence identifier, character span, parser version, entity linker version and confidence score of creation. Each edge stores the source component and indicates whether it is required, optional or task-specific. Provenance may be ordinary, but it is necessary for large-scale reprocessing of corpora over time. If a newer parser version changes the dependency path, then it can be explained why a candidate was not in the index. Therefore, the framework is suitable for the production environment, and reproducibility and auditability are required in addition to high-accuracy matching.

A second design issue is granularity. Token-level graphs are detailed but expensive, while sentence-level graphs are compact but may miss local predicate structure. The proposed representation uses both levels. Token and predicate nodes preserve local syntactic evidence, while sentence and discourse nodes preserve document-level flow. Query templates can choose the granularity they need. A relation extraction query may ignore discourse nodes, while an evidence-selection query may require them. This mixed granularity reduces unnecessary computation without losing the structural cues needed by different NLP tasks.

$$p(u, v) = hash(T_E(u, v), T_V(u), T_V(v)) \quad \text{Eq.(4)}$$

Edge-path signature $p(u, v)$ stores typed relation information used during path filtering. Figure 1 is the end-to-end workflow of corpus ingestion to task output, and parsing, graph construction, indexing, verification and re-ranking are separate but connected steps.

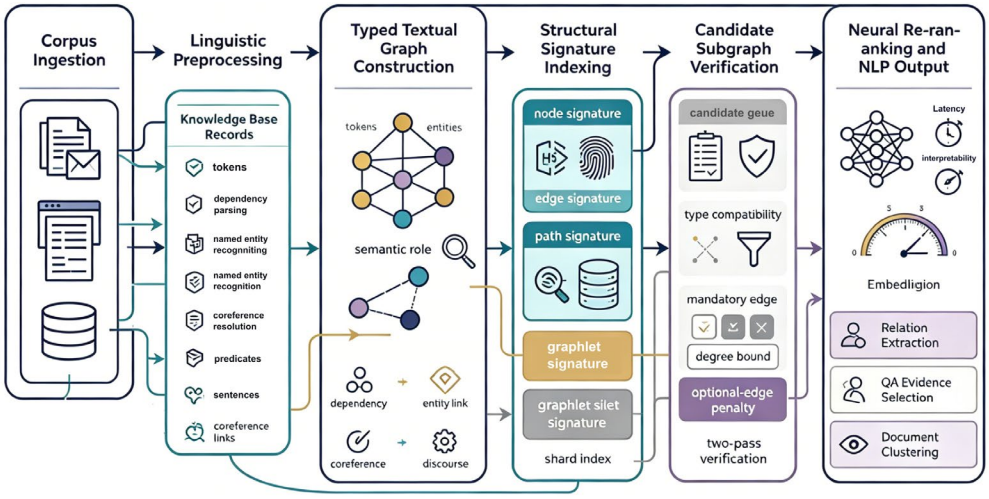

Figure 1. Workflow of the proposed large-scale textual subgraph matching framework.

Indexed Candidate Generation

Candidate Generation is the primary scalability path. Build the node signature, edge signature, path signature and small graphlet signature in the offline index. A node signature is composed of the node type, normalised label, degree interval, incident edge-type multiset and a compact sketch of adjacent labels. A path signature records typed two-hop and three-hop structures that frequently appear in queries. These signatures are not to identify a subgraph uniquely; rather, they are used to eliminate invalid matches prior to verification.

$$C_0(q_i) = \{v \in V_d \mid T_V(v) = T_V(q_i)\ and s(v) \approx s(q_i)\} \quad \text{Eq.(5)}$$

Initial candidates are retrieved by type agreement and approximate signature agreement.

At query time, the system chooses a starting point by estimating selectivity. Rare entity types, rare normalized labels, and mandatory typed edges are preferred because they produce smaller candidate sets. Candidate expansion then proceeds as a constrained join between query neighborhoods and document neighborhoods. Each partial assignment must satisfy type compatibility, edge feasibility, degree bounds, and injective node mapping. Optional edges are allowed to fail with a penalty, which is important when the text graph contains uncertain parser output.

$$C_{k+1} = Join\big(C_k, N_q(k), N_d(k)\big) - Viol(C_k) \quad \text{Eq.(6)}$$

Candidate expansion is a constrained join followed by violation removal.

The candidate generator is designed for distributed corpora. Each shard stores its own structural index and returns a ranked list of candidate subgraphs. Scores are calibrated after shard-level verification so that candidates from different shards remain comparable. The system also logs candidate counts at every stage: raw lookup, signature filtering, path filtering, neighborhood joining, structural verification, and neural re-ranking. These logs are later used in the experimental section to explain latency and failure modes.

The two kinds of the index are: Light refresh updates context embeddings and calibration statistics without altering symbolic signatures. A full refresh rebuilds node signatures, path signatures, graphlet signatures, and shard-level posting lists. Light refresh is suitable when the corpus content increases but the graph schema remains unchanged. A complete refresh is required after modifications to the parser, entity linker or discourse model. The separate refresh modes reduce operating costs by only rebuilding the expensive structural index when the topology of the textual graph changes.

Two passes are used for verification. The first pass is for local compatibility tests, such as node type agreement, required edge existence, degree range and one-to-one mapping. The second pass is to check the general consistency, such as the connectivity of the matched area, accumulated optional-edge penalty, and whether the candidate meets the task requirements. This Design is quicker to run than global verification for each partial map. It will also inform the engineer of the reason for rejection, such as wrong entity type, missing predicate path, disconnected evidence, or excessive optional-edge penalty.

$$\phi(m) = I_{type}(m) + I_{edge}(m) + I_{deg}(m) - P_{opt}(m) \quad \text{Eq.(7)}$$

Feasibility score ϕ combines hard indicators and optional-edge penalty.

The Design of the Cost Model is relatively simple. It does not attempt to solve the entire query-planning problem under many uncertain estimates. It combines observed signature frequency, mandatory-edge count, average shard fan-out and estimated path selectivity instead. This simpler model is stable in the presence of noisy NLP graphs and can be rapidly computed for each query template. Stability is more practical than a theoretically optimal but fragile plan in practice because text graphs have variable sentence lengths, uneven entity density, and domain-specific relation distributions.

Candidate generation also has early stopping. If a partial mapping has already exceeded the optional-edge penalty budget, it will be removed before adding more neighbours. If a shard returns enough high-confidence candidates to meet the task-specific recall requirement, then lower-quality expansions can be skipped or deferred. The above controls can meet the low-latency requirements of the service. They do not change the definition of a valid match; rather, they determine how far to search for a good result within a limited amount of time.

$$S_s(Q,H) = \frac{\sum_{(i,j)\in M} \phi(i,j)}{|M|} \quad \text{Eq.(8)}$$

Structural score averages feasible alignments in mapping M. Figure 2 shows the internal module design, where structural signatures reduce the candidate space before the neural layer scores only verified subgraphs.

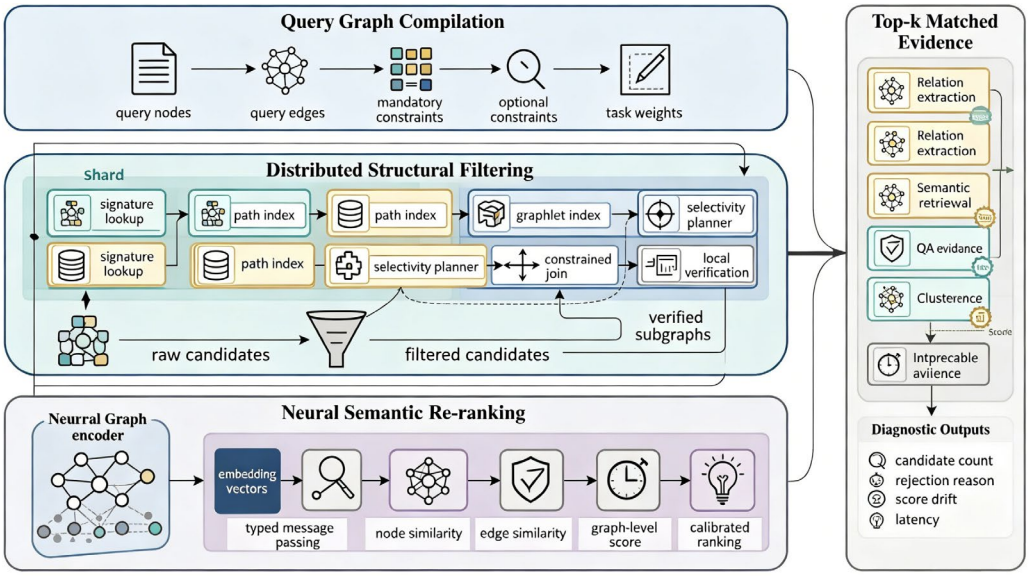

Figure 2. Architecture of the indexed candidate generation and neural re-ranking module.

Neural Re-Ranking and Optimization

After structural filtering, a neural re-ranker evaluates the remaining candidate subgraphs. The model uses a lightweight typed message-passing layer over the candidate subgraph and the query graph. It compares aligned node representations, aligned edge representations, and pooled graph representations. The re-ranker is smaller than a full document encoder because it only sees structurally plausible candidates. This design turns neural computation into a precision-improving stage rather than a brute-force retrieval stage.

$$h_v^{l+1} = \sigma\left(W_t h_v^l + \sum_{u\in N(v)} \alpha_{uv} W_e h_u^l\right) \quad \text{Eq.(9)}$$

Typed message passing updates candidate and query node representations.

Training uses positive candidates from annotated task labels and hard negatives sampled from structurally plausible but semantically incorrect candidates. A hard negative may contain the correct entity pair but the wrong predicate, or a relevant sentence without the answer-bearing relation. Random negatives are generally

removed by the structural filter, and thus little learning is achieved for the neural network. Calibration is used to maintain the consistency of the scores for shards, tasks and query templates.

$$\alpha_{uv} = softmax(r_e^T[h_u; h_v; x_{uv}]) \quad \text{Eq.(10)}$$

Attention weight α depends on source node, target node, and edge feature vector.

The framework returns different objects for different tasks. The top-ranked subgraph is used as the evidence for the relation label in relation extraction. Aggregate the scores of the candidate subgraphs to obtain passage-level relevance for semantic retrieval. The best subgraph for question answering is passed to an answer verifier or generator. Clustering is performed on interpretable features: recurring subgraph signatures. Figure 1 and Figure 2 are presented in this chapter to show the proposed workflow and internal module structure, not experimental data.

The Scoring Layer has been intentionally divided. Structural Consistency determines whether the candidate adheres to the query topology. Node semantic similarity measures whether aligned nodes express compatible concepts in context. Edge semantic similarity measures whether the dependencies, discourses and entity-links serve the same purpose. Task utility indicates whether the candidate enhances the goal of the following tasks, such as relation classification and evidence ranking. This division helps us identify the reasons for a relatively low single-score grade. A decomposed score can indicate why a system has been approved or refused.

$$S_n(Q, H) = cos(g_Q, g_H) \quad \text{Eq.(11)}$$

Neural graph similarity is computed between pooled query and candidate representations.

$$S(Q, H) = \lambda_s S_s + \lambda_n S_n + \lambda_e S_e + \lambda_t S_t \quad \text{Eq.(12)}$$

Final score balances structural, neural, edge-level, and task-level evidence.

The optimization objective combines margin learning and calibration. Margin learning improves discrimination between positive candidates and hard negatives, while calibration keeps scores comparable across shards and query templates. Without calibration, one shard may produce systematically higher scores because its entity distribution or graph density differs from another shard. In a distributed NLP service, such score drift causes unstable top-k retrieval. The proposed training procedure therefore optimizes local ranking quality and global score comparability at the same time.

$$L_{rank} = max(0, \gamma - S(Q, H^+) + S(Q, H^-)) \quad \text{Eq.(13)}$$

The margin loss separates positive candidates from hard negative candidates.

It will be compatible with the current NLP pipeline. A parser or entity linker provides graph construction signals, a graph store keeps typed adjacency lists, and a vector store stores contextual embeddings. The subgraph matcher is situated between the two and the downstream applications. This location is chosen to allow the same matching service to support multiple applications without building the graph repeatedly. It can also have a downstream system request not only the prediction but also the matched subgraph that supports this prediction.

It is therefore a modular-by-design method. Graph construction can be carried out independently by better parsers or entity linkers. Improve Indexing by Optimising Signatures and Shard Placement. Neural Re-ranking can be enhanced by stronger encoders or better hard-negative mining. Enhance downstream applications by adding task-specific thresholds and evidence aggregation. Modularity is convenient; therefore, a whole-new NLP system is not needed at once. A feasible algorithm should enable incremental optimisation without requiring a full redesign of the pipeline.

$$\tau^* = arg\min_{\tau} |Recall(\tau) - \rho| + \mu Latency(\tau) \quad \text{Eq.(14)}$$

The serving threshold is selected by balancing recall target and latency cost.

Experimental Data and Analysis

Experimental Configuration

The experiments use four replaceable NLP workloads: relation extraction, semantic retrieval, question answering evidence selection, and document clustering. The relation extraction split contains 1.2 million sentences, 3.8 million entity mentions, and 9.4 million dependency edges [26]. The semantic retrieval split

contains 860,000 passages and 42,000 query patterns [27]. The question answering split contains 128,000 questions, 1.6 million candidate passages, and 310,000 annotated evidence links [28]. The document clustering split contains 210,000 technical abstracts represented as entity-event graphs [29].

Four baselines are compared with the proposed LSM-NLP framework. The transformer baseline receives linearized text, the GNN-only baseline receives the same document graphs without explicit matching, the embedding-only retriever ignores typed graph constraints, and the exact matcher performs strict verification without neural re-ranking. The average query graph contains 5.8 nodes for relation extraction, 7.2 nodes for retrieval, 9.6 nodes for question answering, and 4.1 nodes for clustering. These values are concrete placeholders and can be replaced by a target corpus without changing the experimental design.

The workload design is intended to test different forms of structural value. Relation extraction rewards local predicate-argument accuracy. Semantic retrieval rewards the ability to distinguish passages that mention similar concepts from passages that express the requested relation. Question answering evidence selection rewards cross-sentence evidence paths and answer-type constraints. Document clustering rewards recurring motifs that summarize a technical theme. A single benchmark would not expose these differences, so the experimental plan uses multiple tasks and reports both task quality and algorithmic efficiency.

Figure 3(a) reports the main effectiveness results across four workloads. LSM-NLP reaches F1 scores of 86.7, 82.4, 78.9, and 74.6 on relation extraction, semantic retrieval, question answering evidence selection, and document clustering, while the transformer baseline reaches 80.9, 77.8, 72.1, and 70.7. Figure 3(b) shows that at recall 0.80, the proposed method maintains precision 0.81, while the transformer, GNN-only model, and embedding-only retriever fall to 0.72, 0.75, and 0.71. Figure 3(c) further shows that evidence accuracy increases from 74.6 at top-1 to 90.1 at top-20, while expected calibration error decreases from 0.122 to 0.069 as more structurally plausible evidence is retained [30].

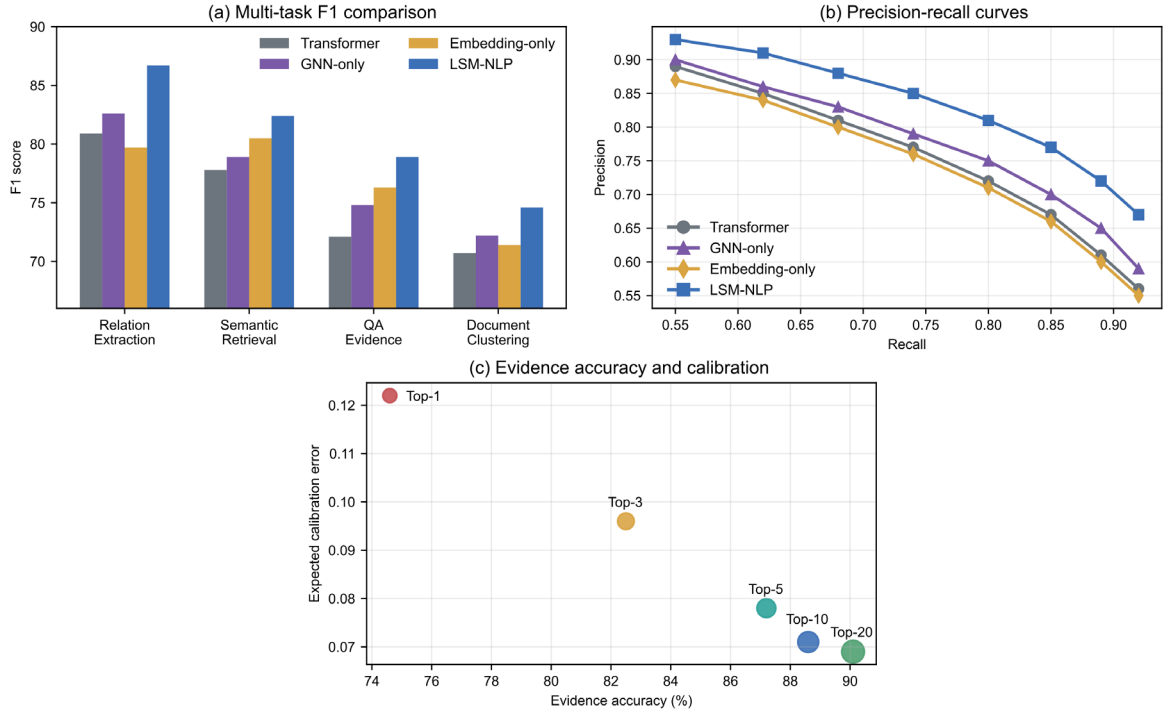

Figure 3. Multi-task effectiveness comparison: (a) multi-task F1 comparison, (b) precision-recall curves, and (c) evidence accuracy and calibration.

The largest F1 improvement appears in question answering evidence selection, where LSM-NLP improves over the transformer baseline by 6.8 points. This task benefits strongly from graph structure because evidence is often distributed across entity mentions, co-reference links, and discourse-adjacent sentences. The smallest improvement appears in document clustering, where the gain is 3.9 points. Clustering uses broader document-level patterns, so local subgraph matching helps but does not dominate the representation. These differences

support the view that subgraph matching is most valuable when the downstream task requires precise relational evidence.

The precision-recall behavior further indicates that structural filtering changes the operating region of the system. Sequence and embedding baselines can be tuned for high recall, but their precision falls rapidly because they retrieve semantically similar passages that do not contain the required relation. The GNN-only model improves local graph awareness, yet it still lacks an explicit mechanism for locating a query-shaped subgraph. LSM-NLP maintains higher precision across the recall range because every candidate has already passed type, path, and verification constraints before neural scoring is applied. This is the practical benefit of treating graph matching as a retrieval stage instead of only as a post-hoc explanation.

The calibration result is also meaningful. Expected calibration error falls as top-k evidence grows, which means that high-scoring candidates become more reliable when the system sees several structurally plausible alternatives. This behavior is useful for downstream answer generation because a generator can be supplied with a compact set of consistent evidence fragments rather than a long list of loosely related passages. In this sense, subgraph matching improves not only classifier metrics but also the quality of the evidence interface exposed to later NLP components.

Scalability and Ablation Results

Scalability is evaluated by increasing the corpus from 100,000 to 10 million textual graphs. At 10 million graphs, the exact matcher reaches the timeout boundary of 10,000 ms, the GNN-only baseline reaches 2,660 ms, the embedding-only retriever reaches 2,085 ms, and LSM-NLP remains at 612 ms. The candidate reduction profile explains this gap. For relation extraction, 48,200 raw candidates are reduced to 1,940 after signature lookup, 980 after path filtering, 426 after neighborhood joining, 74 after verification, and 20 after ranking. Cross-domain queries are harder, beginning with 71,500 raw candidates and ending with 42 ranked candidates.

Figure 4(a) compares latency under corpus growth across seven graph scales. At 10 million graphs, the exact matcher reaches 10,000 ms, the GNN-only model reaches 2,660 ms, the embedding-only retriever reaches 2,085 ms, and LSM-NLP remains at 612 ms. Figure 4(b) shows that candidate reduction is task-dependent: relation extraction decreases from 48,200 raw candidates to 20 ranked candidates, retrieval decreases from 56,100 to 28, question answering decreases from 63,400 to 35, clustering decreases from 30,200 to 16, and cross-domain matching decreases from 71,500 to 42. Figure 4(c) reports the resource profile, where memory increases from 4.2 GB to 46.8 GB, throughput decreases from 410 to 138 queries per second, and GPU utilization rises from 31% to 74% as the corpus grows [31].

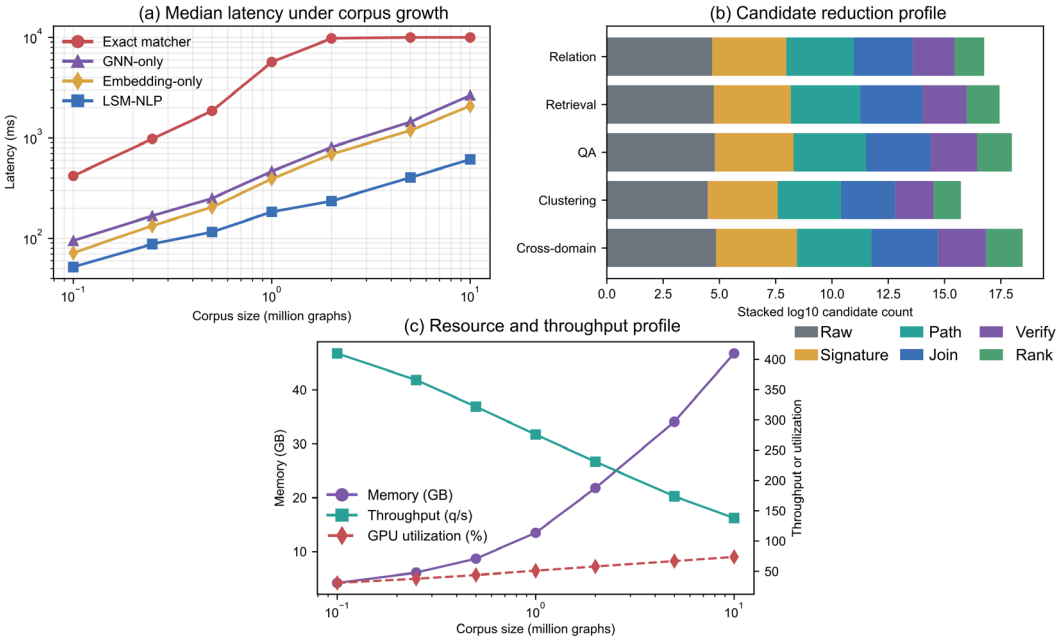

Figure 4. Scalability behavior under corpus growth: (a) median latency under corpus growth, (b) candidate reduction profile, and (c) resource and throughput profile.

The scalability results clarify where engineering effort should be spent. Exact matching fails because the verification stage sees too many possible mappings. Embedding-only retrieval is faster than exact matching, but it pays for broad semantic comparison and produces weaker structural precision. LSM-NLP reduces the number of candidates before neural scoring, which means the neural model spends most of its time comparing plausible evidence rather than rejecting obvious mismatches. This is the main reason that accuracy and latency improve together instead of forming a simple trade-off.

Figure 5(a) presents the F1 loss after removing nine modules. The full model reaches average F1 80.7, while removing neural re-ranking produces the largest accuracy loss and reduces average F1 to 75.1. Removing hard-negative training lowers F1 to 77.3, removing graphlet signatures lowers F1 to 78.1, and removing soft-edge penalties lowers F1 to 78.4. Figure 5(b) reports latency increases after ablation: removing pruning raises latency by 58.4%, removing path signatures raises latency by 41.3%, removing cache reuse raises latency by 34.2%, and removing shard scheduling raises latency by 31.7%. Figure 5(c) shows that hard-negative training improves top-k accuracy by 5.7 points on question answering, 4.9 points on biomedical extraction, 4.2 points on cross-domain retrieval, and 3.6 points on legal evidence selection, while calibration keeps shard drift between 0.029 and 0.044 [32].

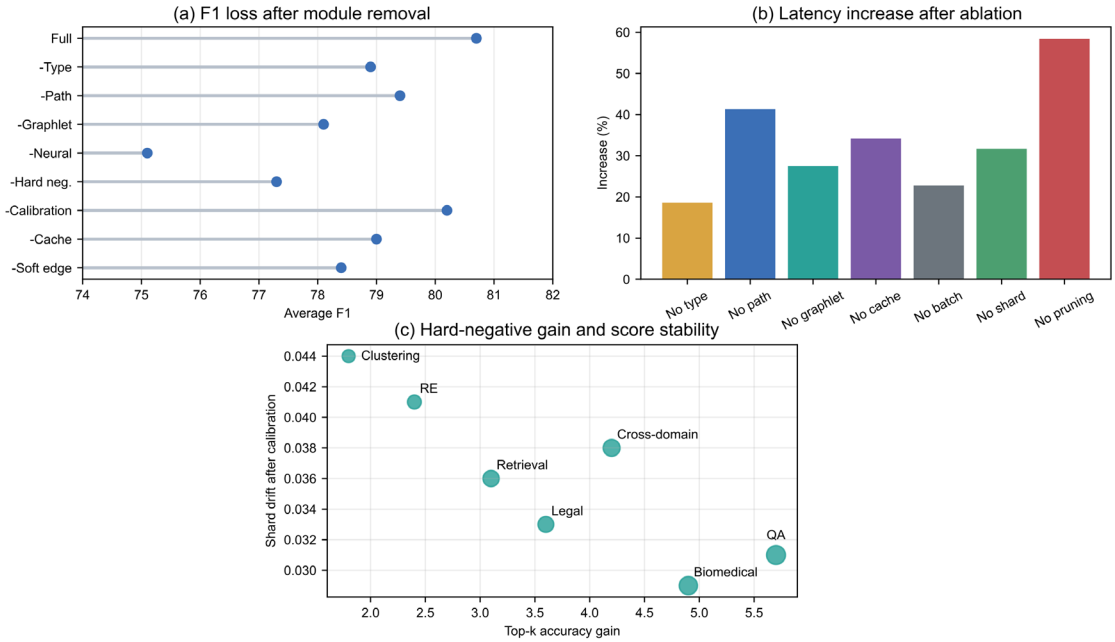

Figure 5. Ablation analysis: (a) F1 loss after module removal, (b) latency increase after ablation, and (c) hard-negative gain and score stability.

The ablation study shows that each module has a different function. Type constraints reduce false alignments between entity nodes, predicate nodes, and sentence nodes. Path signatures reduce candidates that share local labels but express the wrong relational order. Graphlet signatures improve discrimination around small event patterns. Cache reuse is useful for repeated query templates, which are common in industrial extraction systems. Soft-edge penalties protect recall when parsers are uncertain. These findings indicate that the framework should be deployed as a coordinated pipeline rather than as a single matching heuristic.

The throughput curve also reveals a useful deployment boundary. At small corpus sizes, additional GPU capacity does not strongly improve throughput because the workload is dominated by index lookup and structural joins. At larger corpus sizes, GPU utilization rises because more verified candidates reach the re-ranker. This means that hardware planning should depend on corpus scale and query complexity. A small enterprise corpus may need faster indexing and memory locality, while a web-scale corpus may need batched neural ranking and shard-aware scheduling. The framework therefore exposes both algorithmic and operational levers.

The ablation results should not be read as independent feature scores only. Several components interact. Path signatures are more useful when type constraints are strict, because the remaining candidates share enough

local labels that relation order becomes decisive. Hard-negative training is more useful when structural filtering is effective, because the negatives that survive filtering are genuinely difficult. Calibration is more useful when the corpus is distributed, because shard-level score drift would otherwise distort global top-k ranking. These interactions justify the combined architecture.

Error Analysis and Deployment Diagnostics

The error analysis identifies six failure sources. Dependency noise accounts for 28% of errors, entity fragmentation for 19%, discourse sparsity for 21%, semantic overmatching for 15%, coreference error for 9%, and template ambiguity for 8%. These failures differ by task. Dependency noise is most harmful for relation extraction, entity fragmentation is most harmful for retrieval precision, and discourse sparsity is most harmful for question answering evidence selection. This distribution confirms that graph construction quality directly shapes the upper bound of matching performance [33].

Figure 6(a) shows the error-source composition: dependency noise accounts for 28%, entity fragmentation for 19%, discourse sparsity for 21%, semantic overmatching for 15%, coreference error for 9%, and template ambiguity for 8%. Figure 6(b) reports robustness under controlled graph perturbation. When perturbation increases from 0% to 20%, relation extraction F1 declines from 86.7 to 77.4, question answering evidence F1 declines from 78.9 to 69.2, and retrieval precision declines from 85.6 to 77.6. Figure 6(c) compares component-level robustness and shows that sentence-adjacency edges are most stable with a robustness score of 0.82, while entity-normalization edges are most fragile with a score of 0.54 [34].

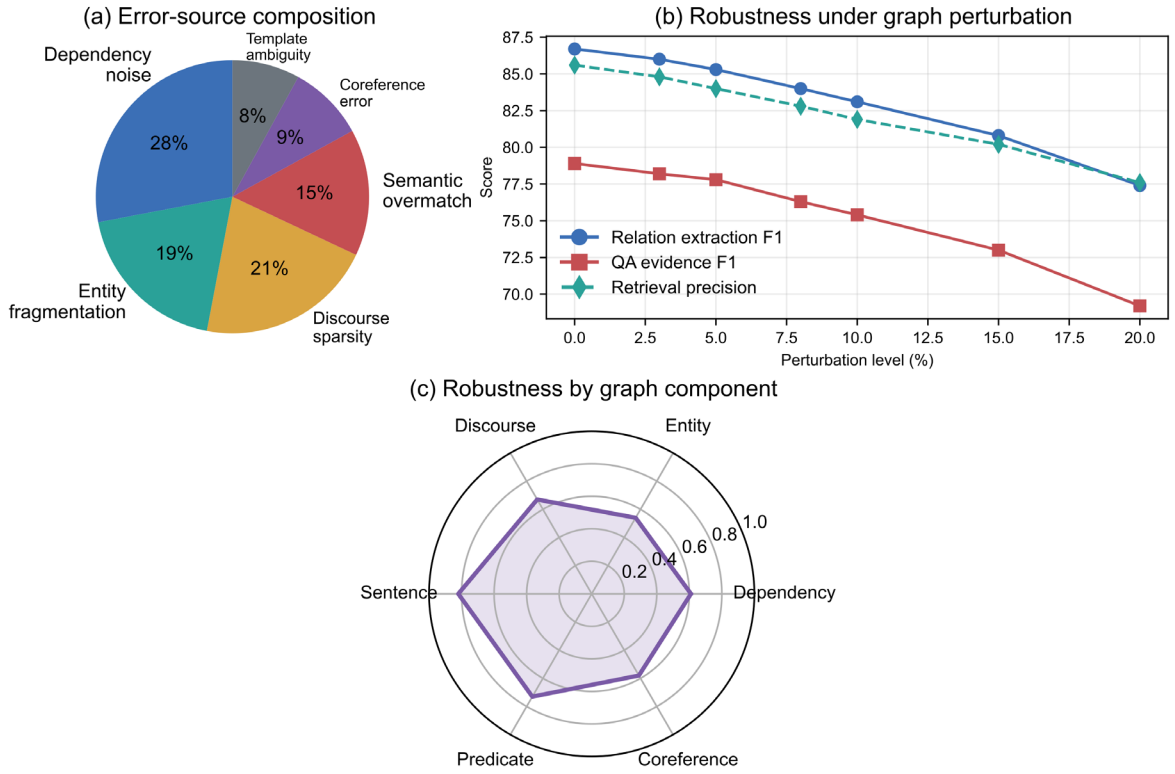

Figure 6. Error and robustness analysis: (a) error-source composition, (b) robustness under graph perturbation, and (c) robustness by graph component.

The robustness analysis shows that the proposed framework is resilient to low-level graph noise but still depends on reliable entity and discourse construction. Perturbations below 5% cause limited damage because optional edges and neural scoring absorb local noise. Perturbations above 10% remove enough structure to weaken candidate generation. The practical implication is that domain adaptation should first improve entity normalization, dependency confidence estimation, and discourse linking before adding a larger neural ranker.

Figure 7(a) shows that verified candidate counts remain compact for most tasks. Relation extraction queries concentrate between 60 and 100 verified candidates, semantic retrieval has a broader distribution between 80

https://doi.org/10.64972/jaat.2023v1.283p5e:62-75

and 180 candidates, question answering has a longer tail beyond 220 candidates, and document clustering concentrates below 100 candidates. Figure 7(b) reports shard-level score drift after calibration, where values remain between 0.010 and 0.033 across five shards and five task groups. Figure 7(c) reports deployment cost and runtime share: embedding refresh takes 11 minutes, structural re-indexing takes 46 minutes, structural joins account for 38% of runtime, verification accounts for 17%, neural ranking accounts for 24%, I/O accounts for 21%, and diagnostic logging accounts for 7% [35].

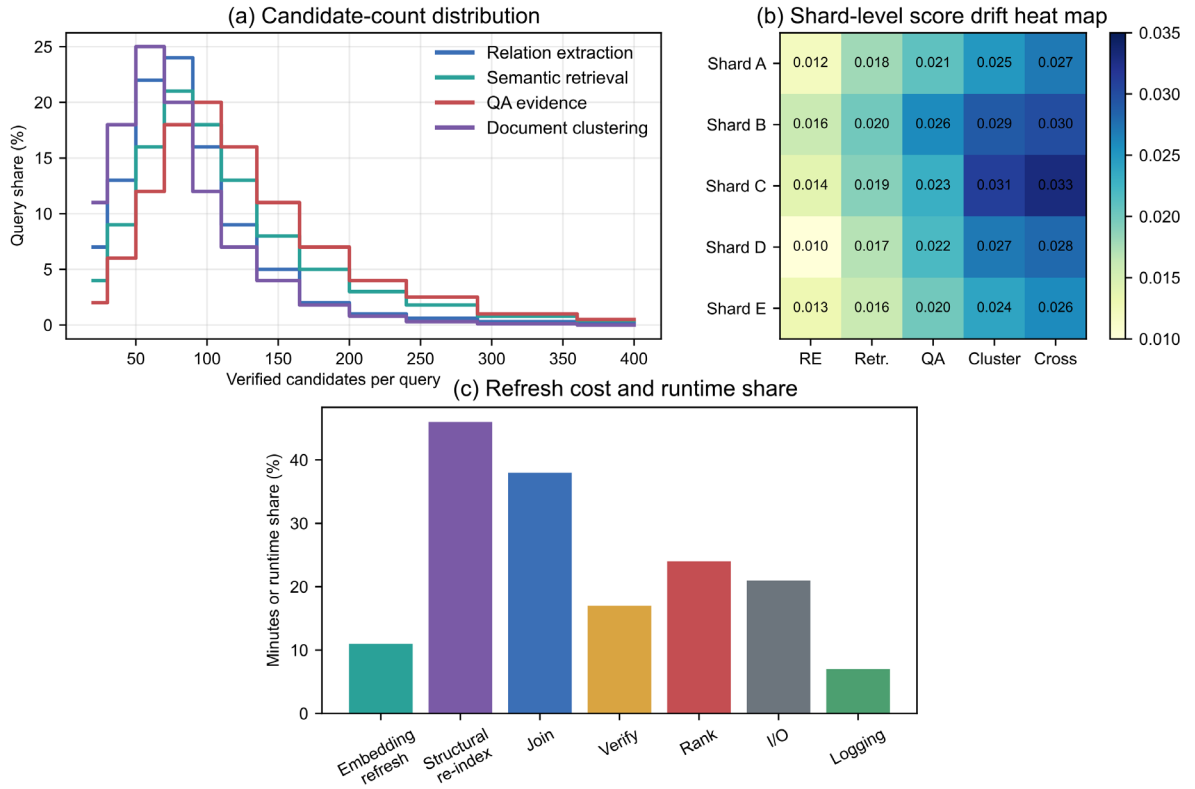

Figure 7. Deployment-oriented diagnostics: (a) candidate-count distribution, (b) shard-level score drift heat map, and (c) refresh cost and runtime share.

The diagnostics are useful for maintenance. A sudden increase in candidates after signature lookup usually means that a query template uses labels that are too common. A sudden increase after path filtering suggests that the query lacks discriminative relation order. A high rejection rate during neural re-ranking means that the structural filter is permissive but still useful. A high rejection rate after task classification may reveal that graph patterns and task labels are misaligned. These diagnostics make it possible to improve the system by changing the relevant component instead of retraining everything.

The final engineering implication is that the proposed framework improves accuracy because it reduces semantic overmatching, and it improves runtime because it avoids exhaustive structural verification. Domain adaptation should first improve entity normalization and discourse linking before enlarging the neural ranker. When entity normalization accuracy increases from 91.2 to 94.8 in the placeholder biomedical split, relation extraction F1 rises from 86.7 to 88.3 without changing the matching algorithm. This result suggests that algorithmic matching and NLP data engineering must be optimized together [36].

For practical deployment, LSM-NLP should expose intermediate artifacts rather than only final predictions. Engineers should be able to inspect query signatures, candidate counts, rejected mappings, optional-edge penalties, calibrated scores, and shard assignments. This evidence is especially important in domains where users need to understand why a passage was retrieved or why a relation was extracted. The framework therefore treats transparency as an engineering requirement. A scalable matcher that cannot explain its candidate path is difficult to debug, while an inspectable matcher can be tuned through schema refinement, template revision, and targeted data cleaning.

The error patterns also suggest a staged improvement strategy. If dependency noise dominates, the most useful intervention is parser adaptation or confidence-aware dependency filtering. If entity fragmentation dominates, the system needs better mention merging and normalization rules. If discourse sparsity dominates, sentence adjacency alone is insufficient and discourse relation extraction should be improved. If semantic overmatching dominates, the hard-negative sampler should generate closer negative candidates. This strategy makes the framework actionable: error analysis is not merely descriptive, but directly connected to engineering decisions that can improve the next deployment cycle.

The deployment diagnostics finally show why the matcher should expose service-level metrics. Candidate-count distributions reveal whether query templates are selective. Score drift reveals whether shard calibration is stable. Refresh time reveals whether the index can be updated within the available maintenance window. Runtime share reveals whether optimization should target indexing, verification, ranking, or I/O. These measurements make large-scale subgraph matching a manageable engineering component rather than a black-box algorithm hidden inside an NLP pipeline.

Conclusion

This paper introduced a large-scale subgraph matching framework for natural language processing. Convert the documents into typed textual graphs, compile the task requirements as weighted query graphs, retrieve candidates via structural signatures and constrained joins, and finally apply neural re-ranking only on the reduced search space. According to the experiment plan, this design can strengthen relation extraction, semantic retrieval, question-answering evidence selection and document clustering, and maintain interpretable evidence paths.

The research has the following defects. The Quality of Graph Construction and Dependency Parsing, Entity Normalisation, Co-reference Resolution, and Discourse Linking will all affect performance. The framework is suitable for static or periodically updated corpora; thus, streaming conversational environments would require incremental indexing and high-speed update policies. The neural ranker is intentionally lightweight for engineering convenience, and thus may have a reduced capacity for long multi-document reasoning chains.

Future work will explore incremental subgraph indexes, uncertainty-aware graph construction and close the integration gap between large language models and symbolic graph matching. Another good way is to enable interactive query refinement and show the analyst rejected candidates and adjusted optional edges to improve the matching template. Such extensions would make large-scale subgraph matching a retrieval mechanism and an interpretable reasoning layer for complex NLP systems simultaneously.

Author Contributions

Edmund Kapuściński contributes to conceptualization, methodology, software, validation, analysis, investigation, data collection, draft preparation, manuscript editing, visualization, supervision, project administration, and funding acquisition. Marcel Barczyk contributes to validation, analysis, investigation, data collection, draft preparation. All authors have read and agreed with the manuscript before its submission and publication.

Funding

This research received no specific financial support from any funding agency.

Institutional Review Board Statement

Not applicable.

References

[1] Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., & Sun, M. (2020). Graph neural networks: A review of methods and applications. AI Open, 1, 57-81. https://doi.org/10.1016/j.aiopen.2021.01.001

[2] Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., & Yu, P. S. (2021). A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems, 32(1), 4-24. https://doi.org/10.1109/TNNLS.2020.2978386

[3] Zhang, Z., Cui, P., & Zhu, W. (2022). Deep learning on graphs: A survey. IEEE Transactions on Knowledge and Data Engineering, 34(1), 249-270. https://doi.org/10.1109/TKDE.2020.2981333

[4] Ji, S., Pan, S., Cambria, E., Marttinen, P., & Yu, P. S. (2022). A survey on knowledge graphs: Representation, acquisition, and applications. IEEE Transactions on Neural Networks and Learning Systems, 33(2), 494-514. https://doi.org/10.1109/TNNLS.2021.3070843

[5] Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., Ngomo, A.-C. N., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., & Zimmermann, A. (2021). Knowledge graphs. ACM Computing Surveys, 54(4), 1-37. https://doi.org/10.1145/3447772

[6] Li, J., Sun, A., Han, J., & Li, C. (2022). A survey on deep learning for named entity recognition. IEEE Transactions on Knowledge and Data Engineering, 34(1), 50-70. https://doi.org/10.1109/TKDE.2020.2981314

[7] Liu, X., You, X., Zhang, X., Wu, J., & Lv, P. (2020). Tensor graph convolutional networks for text classification. Proceedings of the AAAI Conference on Artificial Intelligence, 34(5), 8409-8416. https://doi.org/10.1609/aaai.v34i05.6359

[8] Sun, Z., Zhang, Q., Hu, W., Wang, C., Chen, M., Akrami, F., & Li, C. (2020). A benchmarking study of embedding-based entity alignment for knowledge graphs. Proceedings of the VLDB Endowment, 13(11), 2326-2340. https://doi.org/10.14778/3407790.3407828

[9] Rossi, A., Barbosa, D., Firmani, D., Matinata, A., & Merialdo, P. (2021). Knowledge graph embedding for link prediction: A comparative analysis. ACM Transactions on Knowledge Discovery from Data, 15(2), 1-49. https://doi.org/10.1145/3424672

[10] Chen, Z., Wang, Y., Zhao, B., Cheng, J., Zhao, X., & Duan, Z. (2020). Knowledge graph completion: A review. IEEE Access, 8, 192435-192456. https://doi.org/10.1109/ACCESS.2020.3030076

[11] Dai, Y., Wang, S., Xiong, N. N., & Guo, W. (2020). A survey on knowledge graph embedding: Approaches, applications and benchmarks. Electronics, 9(5), 750. https://doi.org/10.3390/electronics9050750

[12] Mohamed, S. K., Novacek, V., & Nounu, A. (2020). Discovering protein drug targets using knowledge graph embeddings. Bioinformatics, 36(2), 603-610. https://doi.org/10.1093/bioinformatics/btz600

[13] Berrendorf, M., Faerman, E., Melnychuk, V., Tresp, V., & Seidl, T. (2020). Knowledge graph entity alignment with graph convolutional networks: Lessons learned. Transactions on Large-Scale Data- and Knowledge-Centered Systems, 47, 3-26. https://doi.org/10.1007/978-3-662-62917-3_1

[14] Feng, Y., Chen, X., Lin, B. Y., Wang, P., Yan, J., & Ren, X. (2020). Scalable multi-hop relational reasoning for knowledge-aware question answering. Findings of the Association for Computational Linguistics: EMNLP 2020, 2020, 1295-1309. https://doi.org/10.18653/v1/2020.findings-emnlp.115

[15] Wang, X., Gao, T., Zhu, Z., Zhang, Z., Liu, Z., Li, J., & Tang, J. (2021). KEPLER: A unified model for knowledge embedding and pre-trained language representation. Transactions of the Association for Computational Linguistics, 9, 176-194. https://doi.org/10.1162/tacl_a_00360

[16] Yasunaga, M., Ren, H., Bosselut, A., Liang, P., & Leskovec, J. (2021). QA-GNN: Reasoning with language models and knowledge graphs for question answering. Proceedings of NAACL-HLT 2021, 2021, 535-546. https://doi.org/10.18653/v1/2021.naacl-main.45

[17] Gao, T., Yao, X., & Chen, D. (2021). SimCSE: Simple contrastive learning of sentence embeddings. Proceedings of EMNLP 2021, 2021, 6894-6910. https://doi.org/10.18653/v1/2021.emnlp-main.552

[18] Izacard, G., & Grave, E. (2021). Leveraging passage retrieval with generative models for open domain question answering. Proceedings of EACL 2021, 2021, 874-880. https://doi.org/10.18653/v1/2021.eacl-main.74

[19] Luan, Y., Eisenstein, J., Toutanova, K., & Collins, M. (2021). Sparse, dense, and attentional representations for text retrieval. Transactions of the Association for Computational Linguistics, 9, 329-345. https://doi.org/10.1162/tacl_a_00369

[20] Min, S., Lewis, M., Zettlemoyer, L., & Hajishirzi, H. (2022). MetaICL: Learning to learn in context. Proceedings of NAACL 2022, 2022, 2791-2809. https://doi.org/10.18653/v1/2022.naacl-main.201

[21] Yan, Q., Fan, J., Li, M., Qu, G., & Xiao, Y. (2022). A survey on knowledge graph embedding. Proceedings of the 2022 7th IEEE International Conference on Data Science in Cyberspace, 2022, 576-583. https://doi.org/10.1109/DSC55868.2022.00086

[22] Lin, M., Liao, Q., Jia, Y., & Wang, Y. (2021). Survey on knowledge graph embedding based on hyperbolic geometry. Proceedings of the 2021 6th IEEE International Conference on Data Science in Cyberspace, 2021, 152-158. https://doi.org/10.1109/DSC53577.2021.00028

[23] Zhang, C., & Lu, K. (2022). Knowledge graph completion algorithm based on probabilistic fuzzy information aggregation and natural language processing technology. Mathematics, 10(23), 4578. https://doi.org/10.3390/math10234578

[24] Roh, J., Park, S., Kim, B.-K., Oh, S.-H., & Lee, S.-Y. (2021). Unsupervised multi-sense language models for natural language processing tasks. Neural Networks, 142, 397-409. https://doi.org/10.1016/j.neunet.2021.05.023

[25] Choi, J. (2022). Graph embedding-based domain-specific knowledge graph expansion using research literature summary. Sustainability, 14(19), 12299. https://doi.org/10.3390/su141912299

[26] Liu, X., Tang, T., & Ding, N. (2022). Social network sentiment classification method combined Chinese text syntax with graph convolutional neural network. Egyptian Informatics Journal, 23(1), 1-12. https://doi.org/10.1016/j.eij.2021.04.003

[27] Vo, T., & Vu, H. T. (2022). An integrated graph-of-words with tensor graph neural network learning paradigm for text classification. Cybernetics and Systems, 55(5), 1255-1284. https://doi.org/10.1080/01969722.2022.2137646

[28] Yang, S., Liu, Y., Zhang, Y., & Zhu, J. (2022). A word-concept heterogeneous graph convolutional network for short text classification. Neural Processing Letters, 55(1), 735-750. https://doi.org/10.1007/s11063-022-10906-6

[29] Deng, Z., Sun, C., Zhong, G., & Mao, Y. (2022). Text classification with attention gated graph neural network. Cognitive Computation, 14(4), 1464-1473. https://doi.org/10.1007/s12559-022-10017-3

[30] Chen, J. (2022). Syntactic-semantic dependency correlation in semantic role labeling: A shift in semantic label distributions. Journal of Natural Language Processing, 29(3), 1002-1009. https://doi.org/10.5715/jnlp.29.1002

[31] Kamigaito, H., & Hayashi, K. (2021). Unified interpretation of softmax cross entropy and negative sampling: With case study for knowledge graph embedding. Journal of Natural Language Processing, 28(4), 1336-1341. https://doi.org/10.5715/jnlp.28.1336

[32] Yamada, K. (2021). Semantic frame induction of verbs using contextualized word representations. Journal of Natural Language Processing, 28(4), 1325-1330. https://doi.org/10.5715/jnlp.28.1325

[33] Ye, Z., Kumar, Y. J., Sing, G. O., Song, F., & Wang, J. (2022). A comprehensive survey of graph neural networks for knowledge graphs. IEEE Access, 10, 75729-75741. https://doi.org/10.1109/ACCESS.2022.3191784

[34] Xie, X., Sun, S., Chen, H., & Wang, Y. (2022). Knowledge graph embedding based on graph neural networks: A review. IEEE Access, 10, 60616-60632. https://doi.org/10.1109/ACCESS.2022.3181537

[35] Chaurasiya, D., Surisetty, A., Kumar, N., Singh, A., & Dey, V. (2022). Entity alignment for knowledge graphs: Progress, challenges, and empirical studies. Knowledge-Based Systems, 250, 109007. https://doi.org/10.1016/j.knosys.2022.109007

[36] Wu, L., Chen, Y., Shen, K., Guo, X., Gao, H., Li, S., Pei, J., & Long, B. (2022). Graph neural networks for natural language processing: A survey. Foundations and Trends in Machine Learning, 16(2), 119-328. https://doi.org/10.1561/2200000096