Vaialbity study of the use of Graph Neural Networks for Entity Suggestion via Dense Retrieval

Graph Neural Networks for Sematic Entity suggestion

Semester Project PSpring 2023
Group Master Thesis

Aalborg Universitet
Computer Science
Selma Lagerlöfs Vej 300 • DK-9220 Aalborg Øst

Aalborg Universitet
Computer Science
Selma Lagerlöfs Vej 300
DK-9220 Aalborg Øst
http://www.cs.aau.dk

AALBORG UNIVERSITET
STUDENTERRAPPORT

Title:
Graph Neural Networks for Sematic Entity suggestion

Theme:
Vaialbity study of the use of Graph Neural Networks for Entity Suggestion via Dense Retrieval

Project Period:
Spring 2023. semester

Project Group:
Computer Science (IT)
Master Thesis

Participant(s):
Bartal Eyðfinsson Veyhe

Supervisor(s):
Tomer Sagi

Numbered Pages: 118

Date of Completion:
18th October 2023

Abstract:

This thesis presents a comprehensive study on enhancing the accuracy of entity linking in scientific table data by leveraging a knowledge base and machine learning techniques. The research focuses on generating multimodal embeddings for both entity linking and corpus embedding, aiming to improve the suggestion of candidates for entity linking. The study addresses the challenge of ambiguity in entity linking by utilizing a knowledge base for disambiguation and employing a semantic heuristic function to differentiate between overlapping entities. The research methodology involves the use of various datasets, including the Bacteria Biotope datasets, OntoBiotope-NLP dataset, BioTable dataset, Wikidata Subgraph Dataset, and the WDC Schema.org Table Annotation Benchmark dataset. The thesis also explores the architecture of the Mention Encoder, a model that leverages the BERT model with an additional projection head. The experimental setup adopts the closed-world assumption for the knowledge graph and assumes that the Graph Neural Network model is transductive. The results indicate that while the model is unable to retrieve the correct entities, the suggested entities are more semantically in nature as opposed to lexicographic similarity, demonstrating the feasibility of entity suggestion based on dense retrieval. The study concludes that the model requires further fine-tuning to achieve state-of-the-art performance. [1–6]

Contents

1 | Introduction

The primary objective of this research project is to enhance the accuracy of entity linking in scientific table data by leveraging a knowledge base. This will be achieved by investigating the feasibility of employing machine learning techniques to generate multimodal embeddings for both entity linking and corpus embedding. In contrast to the current state-of-the-art methods that heavily depend on lexicographical features, this project aims to exploit the capabilities of a multimodal embedding approach to improve the suggestion of candidates for entity linking. The main focus is to understand how multimodal embedding can be used to extract relevant entities, considering the contextual data within the corpus for Entity Linking with a Knowledge Base.

Entity linking presents a challenge of ambiguity. For instance, the term "Apple" could denote the corporation or the fruit. To address this ambiguity, a knowledge base will be utilized to disambiguate the entities. Entities often possess multiple names and aliases that refer to the same entity. Therefore, it would be more beneficial to identify the entities based on semantic meaning rather than lexicographical similarities, especially when the name being searched for may not exist in the knowledge graph. Several entities can share the same name, making it crucial to consider the context of the entity when performing entity linking. For example, distinguishing between "Apple" the fruit or the company can be challenging. Hence, a semantic heuristic function will be employed to assist in differentiating between overlapping entities.

The structure of this thesis is organized into six chapters. Chapter 1 serves as an introduction to the subject, providing necessary background information and discussing related works that have influenced this project. Chapter 2 delves into the datasets used in the project, specifically focusing on the conversion of these datasets into mention datasets that cover both tabular data and text data for the mentions. This chapter also covers the target knowledge graphs. Chapter 3 presents the model architecture of the project, including the projection head, mention encoder, entity encoder, and the dual encoder architecture. It also discusses the scoring functions that will be utilized. Chapter 4 outlines the experiments that will be conducted in the project and the evaluation metrics that will be used to assess the results. Chapter 5 presents the results of the experiments and provides a thorough evaluation of these results. Finally, Chapter 6 and Chapter 7 concludes the project, summarizing the findings and suggesting potential avenues for future work.

1.1 Introduction

The primary task in research is data generation, which can be both time-consuming and expensive. To address this issue, one approach is to reuse scientific data from related works, which can help alleviate the problem of data collection. Additionally, incorporating additional data sources can enhance the confidence in research findings by combining self-generated data with external scientific data. However, a critical aspect of reusing scientific data is the discoverability and integration of the data.

One challenge in finding relevant scientific data for a research project is that not all terms and publications adhere to the same naming conventions. Therefore, prior knowledge is required to effectively search for scientific data. Furthermore, integrating multiple sources of scientific data necessitates normalization. For example, consider two data sources for measuring the temperatures at different depths in the Culf of Mexico, as shown in Table 1.1 and Table 1.2. These sources use different depth measurment units and temperature unit. To integrating the results requires transforming temperatures into a standardized unit, such as Kelvin, and the distances unit to m.

Depth (m)	**Temperatures** ($\circ$**C**)
0	25.5
30	23
60	21
90	20
120	18
150	17

Table 1.1: *Mean annual water temperatures at different depths in the northen Gulf of Mexico.*

Depth (ft)	**Temperatures** ($\circ$**F**)
0	78
200	70
400	65
600	60
800	57
1000	50

Table 1.2: *Mean annual water temperatures at different depths in the northen Gulf of Mexico.*

Depth (m)	**Temperatures** ($\circ K$)	**Source**
0	298.65	Table 1.1
0	298.70	Table 1.2
30	296.15	Table 1.1
60	294.15	Table 1.1
61	294.26	Table 1.2
90	293.15	Table 1.1
120	291.15	Table 1.1
122	291.48	Table 1.2
150	290.15	Table 1.1
183	288.7	Table 1.2
244	287	Table 1.2
305	283.15	Table 1.2

Table 1.3: *The output scheme for the two data sources.*

Methodologies that aid in the discovery and integration of scientific data include ontology-based data integration (OBDI) and ontology-based data access (OBDA). OBDI and OBDA are information management systems that aim to provide unified and transparent access to data based on a domain model. In OBDI, the system consists of three components: an ontology, a set of data sources, and the mapping between the two. The ontology is a formal description of the domain of interest, expressed in terms of concepts, attributes, relationships, and logical assertions. The data sources are repositories where data concerning the domain are stored. The mapping specifies the correspondence between the data in the sources and the elements of the ontology. The main purpose of OBDI is to enable users to query the data using the elements in the ontology as predicates.

When an organization manages a single data source, the term OBDA is used. OBDA systems also involve an ontology, but the focus is on accessing and querying the data in the source using the ontology as a conceptual model.

The central notion in both OBDI and OBDA is the ontology, which plays a crucial role in reasoning and deriving new facts from the source data. The axioms of the ontology allow for inference and greatly influence the set of answers that the system can compute during query processing.

One of the key aspects of Ontology-Based Data Integration (OBDI) and Ontology-Based

Data Access (OBDA) is the process of identifying the mapping between terms in a schema that describes scientific data and the concepts in an ontology. This process is illustrated in Figures 1.1 and 1.2, where a schema describes the content in each column as depth and temperature with the units m/ft and ∘ C/F, respectively. The output schema, in this case, is represented in Kelvin/meters. The mapping process then associates each concept in the schema to the corresponding concept in the ontologies. OBDI and OBDA can significantly aid in the discovery and integration of multiple scientific data sources. For instance, they enable researchers to query all data sources related to temperature in the Gulf of Mexico without the need to know every methodology for determining the location of the measurement. Furthermore, when integrating data sources, transformations are often required to normalize different concepts. An example of such a transformation is the conversion of Fahrenheit to Kelvin using the equation $K = (F + 459.67) * 5/9$. This transformation ensures that all temperature data is consistent and comparable across different data sources.

However, scientific data often contains more than just a scheme, presenting unique challenges and opportunities when integrating them. One approach is to deconstruct the table into a graph to change the representation of the data into a similar domain as ontologies, treating the data as an instance of a knowledge graph. Entity alignments can then be performed to find a mapping between the data and an ontology. Additionally, the values and content surrounding scientific data can be used to better understand the meaning of a column, even when staying in tabular form.

Existing approaches primarily rely on syntactic information, such as the textual representation of column names, to elicit the meaning of a column by finding the closest syntactic representation in the ontology. However, an increasing number of approaches have started eliciting the semantic meaning of scientific datasets based on the values and content of the data. For example, Wu et al. [2] introduced a dual-encoder/two-tower architecture that measures the semantic similarity between values/mentions in a source document and concepts in an ontology using the dot product. They demonstrated that encoding the semantic information of values can be used as a retrieval mechanism for semantically similar concepts in the ontology. An example of finding semantically similar concepts in Table 1.1 and Table 1.2 is that that feet and meters are semantically similar to the concept of distance units. However, these approaches assume that values are presented in a linear fashion and that the context used to determine semantic information is based on the words surrounding the value. They also represent the semantic information based on the title, subtitle, and description of a concept provided by the ontology.

The current state-of-the-art methods that use a two-tower approach for semantic-based ODBA/OBDI for scientific data have two gaps. Firstly, they assume that values are presented in a linear textual format, encoding semantic information based on left and right context even when dealing with tabular scientific data. This approach does not take advantage of the implicit structural information present in a tabular structure. Secondly, these methods do not leverage the structural information present in an ontology.

In this thesis, we will explore the use of a graph-based embedding method for the ontology side and a table-based embedding for the dataset side to improve the quality of schema mapping of scientific datasets to ontologies.

1.2 Contribution of the paper

- We propose a dual encoder model for entity suggestion and retrieval based on dense retrieval.
- We propose a training strategy for join-mention embedding and ontologies graph embedding.
- We showcase fine-tuning of the model based on the entity linking task.

1.3 Background

The following sections delve into the intricacies of various concepts and techniques in machine learning and data integration, which are pivotal to the understanding and development of my thesis. The first section explores Graph Neural Networks (GNNs) and Graph Convolutional Networks (GCNs), highlighting their potential in handling complex systems and non-Euclidean data structures, and the challenges they face, such as class imbalance in training datasets [7]. The subsequent section discusses Dimensionality Reduction, a technique used to combat the curse of dimensionality, and various methods employed for this purpose, including the use of neural networks and the encode-decode architecture [7]. The third section introduces Dense Retrieval, a modern approach in information retrieval that leverages deep learning techniques and dense vector representations for more accurate and faster information extraction [8].

The fourth section provides an overview of Knowledge Graphs, a tool for data integration that uses graph structures to represent and reason about knowledge, and their utility in data integration [9]. The fifth section discusses Ontologies, a structured framework for organizing and interpreting diverse data sets, and their role in data integration [10]. The sixth section delves into Schema Matching, a process crucial in Ontology-based data integration, and the challenges it presents [10]. The seventh section discusses Entity Linking (EL), a task in Natural Language Processing (NLP) that involves identifying and linking mentions of real-world entities in a text to their corresponding entries in a knowledge base, and its significance in data integration [10].

The final section discusses Ontology-based Data Integration (OBDI), a method that uses ontologies to unify data from various sources, and the role of entity linking and suggestion in enhancing its effectiveness [9]. These sections collectively provide a comprehensive understanding of the various techniques and concepts that underpin my thesis.

1.3.1 Graph Neural Networks

Graph Neural Networks (GNNs) have emerged as a powerful tool for machine learning tasks on graph-structured data, offering a unique approach to handle complex systems [11]. One of the key components of GNNs is the Graph Convolutional Networks (GCNs), which are designed to handle the non-Euclidean data structure of graphs [12]. GCNs leverage the graph structure to define convolution operations in the graph domain, enabling the extraction of local and global features from graph data.

However, the application of GNNs and GCNs often encounters challenges. One such challenge is the class imbalance problem, where the training dataset is small or the distribution of classes is skewed [7]. This can lead to a model that is biased towards the majority class, thus affecting the performance of the model.

To address these issues, GNNs employ techniques such as node embedding and message passing. Node embedding is a process that maps nodes in a graph to a low-dimensional space, capturing the structural and feature information of the nodes [13]. Message passing, on the other hand, is a mechanism that allows information exchange between nodes, enabling the model to aggregate features from neighboring nodes [14]. This neighboring aggregation is crucial in GNNs as it allows the model to capture the local graph structure and the interactions between nodes.

In conclusion, while GNNs and GCNs present promising solutions for learning on graph-structured data, they also pose challenges that need to be addressed to fully exploit their potential.

1.3.2 Dimensionality Reduction

The majority of machine learning models today are suffering from the curse of dimensionality, where a large number of features are used as the input to a model making the growth of the data requirements exponential for effectively discerning patterns between the input functions and the resulting output [15]. To deal with the increasing amount of features, Dimensionality Reduction techniques have been implemented, with increasing success.

In modern machine learning techniques, dimension reduction is based on neural networks that take in a high dimensional vector and map it to a lower dimensional manifold, where "similar" inputs are mapped to output points that are close to each other [15].

The encode-decode architecture (see Figure 1.1) is a type of neural network that is used for dimensionality reduction. The encoder part of the network is used to map the input to a lower dimensional manifold, and the decoder part of the network is used to map the lower dimensional manifold back to the original input. The encoder part of the network is trained to learn a representation of the input that is useful for the decoder part of the network to reconstruct the original input. The decoder part of the network is trained to reconstruct the original input from the lower dimensional manifold. The loss function is used to measure the difference between the original input and the reconstructed input. The loss function is then used to update the weights and biases of the network.

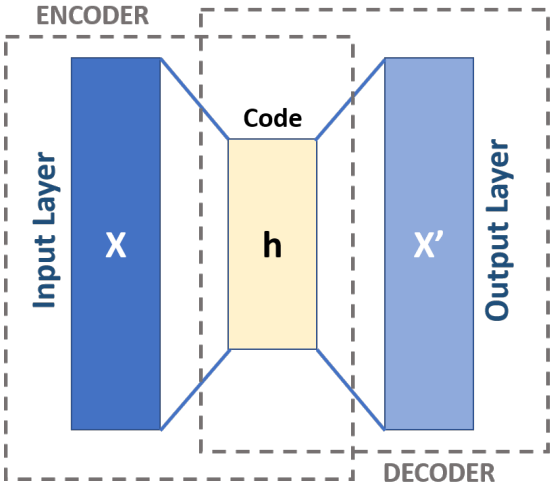


Figure 1.1: *Autoencoder architecture.*

For a given model, where $\mathbf{v} \in V \subseteq \mathbb{R}^n$ is the domain, then the autoencoder model can be defined with two functions $enc : V \to \mathbb{R}^k$ and $dec : \mathbb{R}^k \to \mathbb{R}^n$ where dec is the inverse to

enc. When using neural networks as estimators of the encoder and decoder function, the training paradigm is to take the training data $V^* \subseteq V$ and then for each $\mathbf{v_i} \in V^*$ take $\mathbf{y_i} = dec(enc(\mathbf{v_i}))$ then for a given similar measure $S : \mathbb{R}^n \to \mathbb{R}$ between two inputs.

The dimensionality reduction techniques in an autoencoder come from the output of the *enc* where the dimension of the range is significantly lower than the input dimension.

Further, the Siamese neural network, dual encoders or two towers, and the triplet loss are other techniques used for dimensionality reduction. The Siamese neural network involves two identical neural networks, each taking one of the two input vectors. The last layers of the two networks are then fed to a contrastive loss function [15]. The dual encoders or two towers technique involves two separate encoders for the two different types of inputs. The outputs of the two encoders are then compared to calculate the loss [15]. The triplet loss technique involves three inputs, an anchor, a positive of the same class as the anchor, and a negative of a different class. The goal is to bring the anchor and positive closer together and push the anchor and negative further apart in the embedding space [15]. These techniques aim to move similar objects close together in the manifold and dissimilar objects far apart, thereby reducing the dimensionality of the data.

1.3.3 Dense Retrieval

Dense retrieval is a modern approach in information retrieval that leverages deep learning techniques to retrieve relevant information from a database [16]. Unlike traditional methods that rely on sparse representations, dense retrieval utilizes dense vector representations, often achieved through dimensionality reduction or embeddings [17]. This relationship to embeddings allows for a more nuanced understanding of the data, as it can capture semantic relationships between different pieces of information [8].

The benefits of dense retrieval are manifold. Firstly, it enables semantic search, which allows for a more accurate retrieval of information based on the meaning of the query, rather than just keyword matching [16]. Secondly, dense retrieval is faster than traditional methods, as it can calculate the target embeddings offline and only needs to calculate the query embeddings at runtime [17].

However, dense retrieval is not without its drawbacks. The main disadvantage is that it requires a significant amount of computational resources, both for training the model and for calculating the embeddings [8]. Furthermore, the quality of the retrieval is highly dependent on the quality of the embeddings, which can be influenced by factors such as the choice of model and the quality of the training data [16].

1.3.4 Knowledge Graphs: A Comprehensive Overview

Knowledge graphs, a potent tool for data integration, offer a structured and semantically rich framework for organizing and linking data from diverse sources. They have found extensive applications in various fields such as information retrieval, natural language processing, and data mining. Essentially, a knowledge graph is a database type that employs graph structures to represent, integrate, and reason about knowledge. It comprises entities (nodes) and relationships (edges) between them, where entities symbolize real-world objects or concepts, and relationships depict the connections between these entities (*see Figure 1.2*).

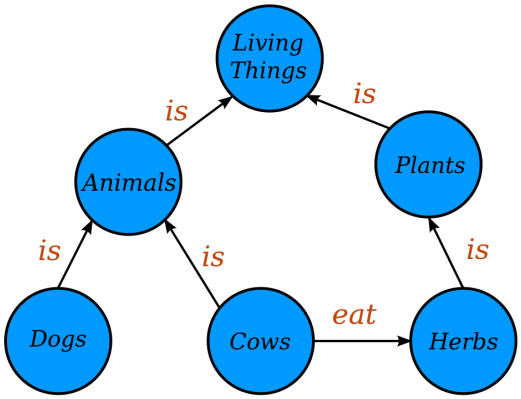


Figure 1.2: *An example of a knowledge graph*

In the context of a knowledge graph, relations and classes are defined using formal logic. Relations serve as predicates linking two entities, representing a specific connection type between them. Conversely, classes are sets of entities sharing common properties, defined by a set of conditions that an entity must satisfy to be a class member. Instances of classes in a knowledge graph are specific entities belonging to a particular class. For instance, in a knowledge graph about animals, "Lion" could be an instance of the class "Mammal", implying that "Lion" satisfies all conditions defining the class "Mammal".

Knowledge graphs prove particularly useful for data integration as they provide a flexible and expressive framework for linking and organizing data from diverse sources. They allow for the representation of complex relationships between entities, and their graph structure facilitates easy querying and information retrieval. Moreover, their use of formal logic enables automated reasoning and inference, which can help uncover new insights from the integrated data.

1.3.5 Ontologies

Ontologies are a key component in the field of data integration, providing a structured framework for organizing and interpreting diverse data sets. They are essentially a formal way to describe things and the relationships between them, making it possible to process data with automated tools.

An ontology can be visualized as a graph, where the nodes represent entities or concepts, and the edges represent the relationships between these entities. For example, in a medical ontology, the nodes might represent diseases, symptoms, and treatments, while the edges might represent relationships like "is a symptom of" or "is a treatment for".

In terms of formal logic, ontologies are typically defined using a set of classes and relations. A class is a collection of entities that share common properties or characteristics. For instance, in a medical ontology, "Disease" might be a class that includes entities like "Cancer" or "Diabetes". Relations, on the other hand, define how classes or entities are related to each other. For example, the relation "is a symptom of" might connect the class "Symptom" to the class "Disease".

Instances of classes in an ontology are specific entities that belong to a class. For example, "Breast Cancer" might be an instance of the class "Disease". Each instance can have properties that further describe it, such as "has symptom" or "has treatment".

Ontologies are particularly useful for data integration because they provide a common

vocabulary for describing data, making it easier to combine data from different sources. They also provide a way to infer new information from existing data, by applying the rules and relationships defined in the ontology. This can help to uncover hidden patterns or insights in the data, enhancing the value of data integration efforts.

1.3.6 Schema Matching

Schema matching is the process of identifying and establishing correspondences between the schemas of different data sources. It involves determining the similarities and differences between the structures, attributes, and relationships of the schemas [10].

In the context of Ontology-based data integration, schema matching is crucial. Ontology-based data integration aims to integrate data from multiple sources by mapping them to a common ontology. However, the schemas of these sources may vary significantly, making it challenging to establish mappings between them.

Schema matching helps in addressing this challenge by providing techniques and algorithms to identify and align similar concepts and relationships across different schemas. By matching the schemas, it becomes possible to create mappings between the data sources and the common ontology.

These mappings enable the integration of data from diverse sources into a unified representation, facilitating seamless querying, reasoning, and analysis. Ontology-based data integration leverages schema matching to bridge the gap between heterogeneous data sources and the common ontology, enabling effective integration and utilization of the data.

1.3.7 Entity Linking

Entity Linking (EL) is a pivotal task in the realm of Natural Language Processing (NLP). It involves the detection of mentions of real-world entities within a text and associating them with their corresponding entries in a knowledge base [18]. EL is a fundamental process in data integration, as it enables the consolidation of information from diverse sources, thereby augmenting the comprehensibility and usability of the data [19].

To understand EL, it is essential to comprehend its steps and significance. The process of EL involves three primary steps: entity detection, entity disambiguation, and entity suggestion. Entity detection identifies the mentions of entities in the text. Entity disambiguation resolves the ambiguity of these mentions by linking them to the correct entities in a knowledge base. Entity suggestion is a crucial step in EL as it recommends potential entities that a mention could be linked to, thereby facilitating the disambiguation process.

EL is particularly beneficial in data integration due to its ability to disambiguate and connect entities. In a world where data is often siloed and unstructured, EL provides a means to structure this data and link it to relevant entities, thereby creating a more coherent and integrated data landscape [20]. Furthermore, EL aids in the reduction of data redundancy, as it ensures that the same entity is not represented multiple times in the dataset. This not only improves the efficiency of data storage and retrieval but also enhances the accuracy of data analysis [21].

1.3.8 Ontology-based Data Integration

Ontology-based data integration (OBDI) is a method that uses ontologies to combine data from diverse sources into a unified view. Ontologies, which are formal representations of

knowledge as a set of concepts within a domain, play a crucial role in EL. They provide a structured framework that aids in the disambiguation and linking of entities, thereby enhancing the effectiveness of EL. [9]

The benefits of OBDI are manifold. It facilitates semantic interoperability, improves data quality, and enables more effective data search and retrieval. However, it also presents several challenges, such as the complexity of ontology development and maintenance, and the difficulty of achieving accurate and efficient entity linking and suggestion.

Entity linking and suggestion are vital for OBDI. They enable the accurate mapping of data to the correct entities in the ontology, thereby facilitating effective data integration. However, they also present challenges, such as the difficulty of disambiguating entity mentions and the computational cost of linking entities.

Semantic search can aid in addressing these challenges. By leveraging the semantic relationships in the ontology, it can improve the accuracy and efficiency of entity linking and suggestion, thereby enhancing the effectiveness of OBDI.

1.4 Related work

This section provides an overview of the related works that have significantly contributed to the field of dual encoder architecture for learning entity representation, a crucial aspect of entity disambiguation. The works of He et al. [22], Wang et al. [23], Wu et al. [2], and Louis et al. [24] are discussed in detail, highlighting their unique approaches and methodologies. These studies have employed various techniques for mention encoding and entity embedding, and have used different similarity measures. This thesis aims to build upon these works by integrating a BERT for mention encoding and a Graph Neural Network (GNN) for entity embedding. The similarity measure will be determined using a dot-product, and dense retrieval will be employed. A comprehensive comparison of the four papers and their approaches is provided in Table 1.4. The subsequent sections will delve deeper into the specifics of our proposed methodology and its potential advantages over the existing techniques.

The study of dual encoder architecture for learning entity representation, which is vital for entity disambiguation, was first proposed by He et al. [22]. Their architecture incorporated an auto-encoder on both sides of the dual encoder, with the similarity measure being determined using a dot-product. In contrast to their use of auto-encoders for both entity and mention embeddings, our work will employ a Graph Neural Network (GNN) for entity embeddings and a BERT for mention embeddings. We will update the mention encoder to a BERT encoder, and the entity encoder to a BERT with a GNN encoder. Furthermore, a projection layer will be used to map the two encoders into the same space, so that the dot product can be used for similarity measurement.

Wang et al. [23] furthered this research by exploring a dual encoding system for linking values and entities in a knowledge graph. They used a conditional probability estimator for similarity measurement, while we will use a dot product, so that entities suggestion based on semantic performances can be achieved. A projection layer will be used to map the two encoders into the same space, so that the dot product can be used for similarity measurement.

In a more recent study, Wu et al. [2] utilized a dual encoder architecture for entity suggestion and retrieval based on dense retrieval. They employed BERT for both the

mention and entity encoding and used a dot product for similarity measurement. However, for the entity embedding, they did not use a model that takes the structural information into account. Our work will use a GNN to embed entities. A projection layer will be used to map the two encoders into the same space, so that the dot product can be used for similarity measurement.

Louis et al. [24] proposed a dual-encoder model specifically designed for the retrieval of statutes. The model is fundamentally built upon the principles of query embedding and hierarchical article encoding, both of which are facilitated by BERT-based encoders. In addition to these, a Graph Neural Network (GNN) model is incorporated to augment the article embedding process. This augmentation is realized by harnessing a legislative graph, wherein the article embedding serves as the node features, and is further enhanced by the query embeddings. Unlike their focus on the retrieval of statutes, we will focus on entity linking. They have a separate process for generating the features for the nodes in the graph. Our work investigates integrating the GNN and BERT encoder into a single model. Our work will use a projection layer to map the two encoders into the same space, so that the dot product can be used for similarity measurement.

Table 1.4 provides a comparison of the four papers and their approaches. This thesis will build upon these works by using a BERT for mention encoding and a Graph Neural Network (GNN) for entity embedding. The similarity measure will be determined using a dot-product, and dense retrieval will be employed.

Table 1.4: *Comparison of the four papers and their approaches.*

	Mention encoding	Entity embedding	Similarity measure	Dense retrieval
He et al.	auto-encoder	auto-encoder	Dot-product	✗
Wang et al.	word2vec	TranT	CPD	✗
Wu et al.	BERT	BERT	Dot-product	✓
Louis et al.	BERT	BERT + GNN	Dot-product Eclu-dian similarity	✓
Ours	BERT	GNN	Dot-product Eclu-dian similarity	✓

2 | Datasets

This chapter provides an overview of various datasets and tasks used in the field of biomedical research and data management. It begins by discussing the Bacteria Biotope (BB) datasets, introduced by Bossy et al. [1], which were part of the BioNLP open shared task in 2019. The tasks associated with these datasets are divided into entity detection and relationship extraction, with the aim of identifying entities and relationships within the corpus. The chapter also introduces the OntoBiotope-NLP (OBNLP) dataset, an ontology describing the habitats of microorganisms.

The focus then shifts to the Table Annotation task, which is crucial for understanding and interpreting data in a table. This task includes four distinct subtasks: Column Type Annotation (CTA), Cell Entity Annotation (CEA), Column Property Annotation (CPA), and Table Topic Detection. The chapter discusses the use of various datasets for table annotation, including the BioTable dataset, which is specifically tailored to the biomedical domain. The chapter also explores the challenges and potential biases in the distribution of CTA targets within the BioTable dataset.

The chapter further discusses the Wikidata Subgraph Dataset, a comprehensive knowledge graph used for training models. The complexity and connectivity of an ontology, inferred from the size of its neighbors, is considered when generating an embedding for a node in Graph Neural Networks (GNNs).

Finally, the chapter introduces the WDC Schema.org Table Annotation Benchmark (SOTAB) dataset, a collection of tables used for training purposes. This dataset, along with the Schema.org ontology, is used to annotate tables with entities and properties, despite the imbalance in the frequency of entities in the training dataset.

2.1 Exploration of Bacteria Biotope Datasets in BioNLP 2019

The Bacteria Biotope (BB) datasets, as introduced by Bossy et al. [1], were made available as part of the BioNLP open shared task in 2019. The tasks were bifurcated into two segments: entity detection and relationship extraction. The entity detection task necessitated the identification of entities within the corpus and the discovery of corresponding entities in the NCBI taxonomy database [1] and OntoBiotope habitat [1]. The relationship extraction task was designed to identify ”Lives_In” and ”Exhibits” relationships between mentions in the corpus. The corpus is composed of PubMed titles and abstracts pertaining to microorganisms and extracts from full-text articles related to microorganisms inhabiting food products.

2.1.1 Analysis of Mentions Dataset Statistics

To identify potential issues in the BB dataset, we will scrutinize its statistics. The BB dataset is partitioned into a training and a validation dataset, with the training dataset further divided into a training and a test dataset. The training dataset is utilized to train the model, the test dataset is employed to evaluate the model during training, and the validation dataset is used to assess the model post-training to ensure that the model has not overfitted to the training dataset.

	Training	Validation
Number of mentions	1571	881
Unique entities	299	181

Table 2.1: *BB dataset statistics*

Figure 2.1 illustrates the frequency of the entities in the BB dataset. The most prevalent entity is *OBT:001480* (Cheese) with $n = 133$, followed by a sharp drop-off. This suggests a class imbalance in the training dataset, as discussed in section 1.3, which can cause the training to favor classes that are over-represented in the dataset.

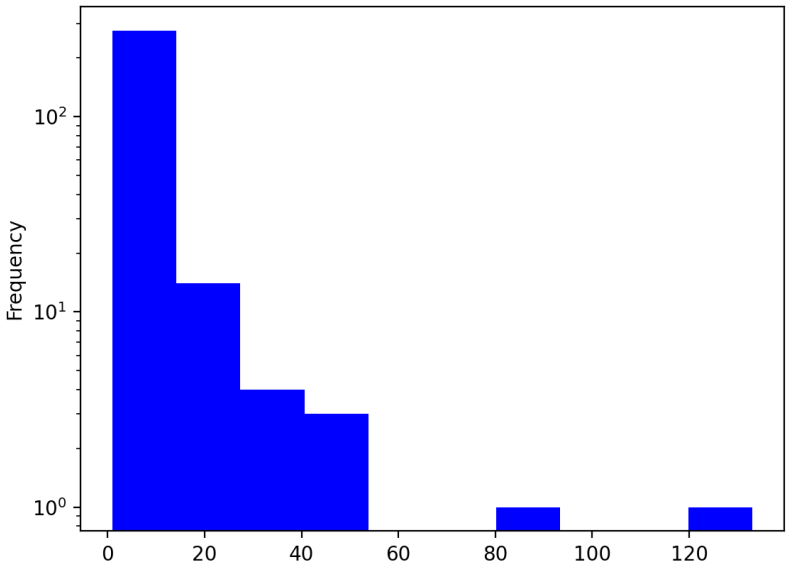


Figure 2.1: *Histogram of entity frequency in the BB training dataset, with the y-axis on a log scale.*

The subsequent section will convert the statistics of the target ontology OntoBiotope-NLP.

2.1.2 Inspection of OntoBiotope-NLP Dataset Statistics

The OntoBiotope-NLP (OBNLP), created for the BioNLP-OST 2019 competition, is an ontology describing the habitats of microorganisms. Each concept in the ontology has a

name, and some concepts also contain a list of synonyms for the given concept [25]. The OBNLP dataset is saved in the obo data format [26], a human-readable text format for ontologies used by OntoBiotope ontology.

	OBNLP
Number of concepts	4219
Names	4219
Synonyms	689
Number of is_a edges	4639

Table 2.2: *OBNLP dataset statistics*

The OBNLP dataset presents one type of relationship in the ontology, the *is_a* type. As shown in Table 2.2, the OBNLP ontology contains 4639 is_a relationships. With the inverse relationships added, the dataset contains a total of 9278 relationships, meaning that the OBNLP ontology contains a total of 9278 triplets.

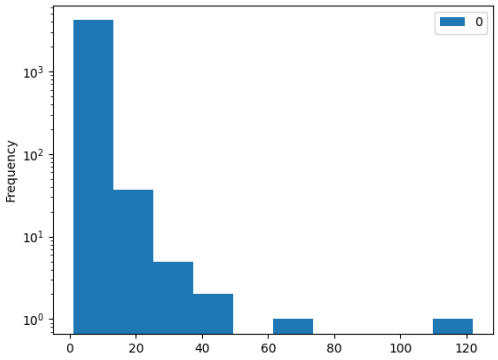


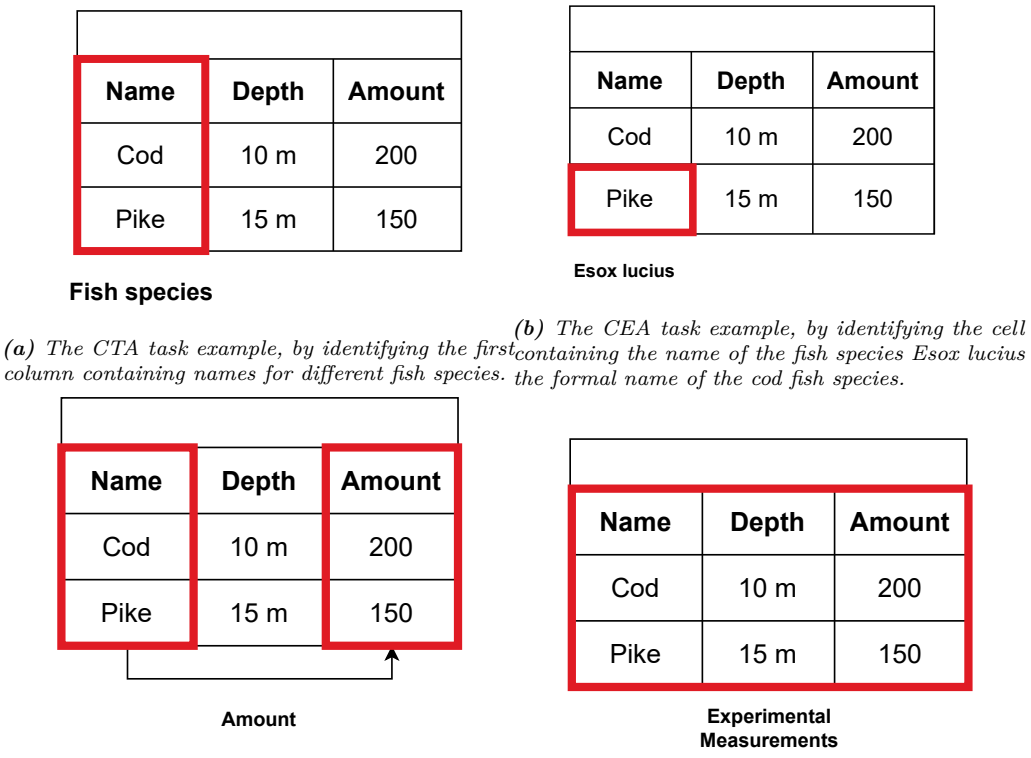
Figure 2.2: *Histogram showing the size of all neighbourhoods, the OntoBiotope-NLP ontology*

The size of the neighbors can provide insights into the complexity and connectivity of the ontology. The histogram in Figure 2.2 shows the distribution of neighborhood sizes in the OntoBiotope-NLP ontology.

2.2 Table Annotation Task

The focus of this research is the Table Annotation task, a complex process that encompasses four distinct tasks. The first task, referred to as Column Type Annotation (**CTA**), involves assigning a semantic type to a column. This task is crucial as it provides a context for the data contained within the column, thereby facilitating a more accurate interpretation of the data. The second task, Cell Entity Annotation (**CEA**), involves assigning a semantic type to a cell. This task is equally important as it provides a specific context for the data contained within the cell. The third task, Column Property Annotation (**CPA**), involves assigning a knowledge graph property to a cell. This task is essential for linking the data

in the cell to a broader knowledge graph, thereby enhancing the utility of the data. The final task, **Table Topic Detection**, involves identifying the overall topic of the table. This task is crucial for understanding the broader context of the table and for facilitating the integration of the table into a larger dataset or analysis. These tasks are illustrated in Figure 2.3, which provides a visual representation of the Table Annotation task and its constituent tasks.



(a) *The CTA task example, by identifying the first column containing names for different fish species.*

(b) *The CEA task example, by identifying the cell containing the name of the fish species Esox lucius the formal name of the cod fish species.*

(c) *The CPA task example, by identifying the relationship between the first and third column, which is the relationship between the fish species and the amount of the fish species.*

(d) *The TTA task example, by identifying the topic of the table, which is Experimental Measurements.*

Figure 2.3: *Examples of the table annotation tasks, From a fictive dataset of fish couth in the Faroe island between the islands.*

A variety of datasets are currently available for table annotation, as outlined in Table 2.3. Among these, BioTable stands out as one of the few that are domain-specific and manually annotated.

BioTable is a unique dataset in the sense that it is specifically tailored to the biomedical domain. This specificity allows for a more focused exploration of the challenges and opportunities associated with table annotation in this field. Furthermore, the manual annotation process employed in the creation of BioTable ensures a high degree of accuracy and reliability, making it an invaluable resource for researchers in the field of biomedical NLP.

Name	Year	Table Count	CEA	CTA	CPA	TTD	Target
T2Dv2 [27]	2017	779	✗	✓	✓	✓	DBpedia
BioTable [28]	2021	110	✓	✓	✓	✗	Wikidata
Tough Tables [29]	2020	180	✓	✓	✓	✗	DBpedia, Wikidata
GitTables [30]	2021	1 Million	✓	✓	✓	✓	DBpedia, Schema.org
WikipediaGS [31]	2019	485 096	✓	✓	✗	✗	DBpedia
VizNet-Sato [32]	2019	78 733	✗	✓	✗	✗	DBpedia
TNCR [33]	2021	9428	✗	✗	✗	✗	✗
WikiTables-TURL [34]	2020	406 706	✓	✓	✓	✗	DBpedia
WDC SOTAB [35]	2023	48 379	✗	✓	✓	✗	Schema.org

Table 2.3: *Table Annotation benchmark and challenge datasets*

This thesis focuses on the task of **Column Type Annotation (CTA)**, a crucial aspect in the field of data management and knowledge representation. The primary objective of CTA is to assign a semantic type to a column in a given dataset. This process is instrumental in enhancing the understanding and interpretation of the data, thereby facilitating more effective data analysis and knowledge extraction.

2.2.1 BioTable dataset statistics

An examination of the CTA targets within the BioTable dataset, as depicted in Table 2.4, reveals that there are only seven distinct entities serving as CTA targets. The frequency of each target varies, with some appearing in all 110 tables (e.g., Q14860489, Q2996394, Q5058355, Q7187, and Q8054), while others appear less frequently (e.g., Q16521 and Q35120).

The uneven distribution of CTA targets in the BioTable dataset could potentially skew the model's learning process, leading to a bias towards targets that appear more frequently. This bias could significantly impact the model's performance, making it crucial to consider these factors during the training phase. Moreover, the limited amount of data available for training further exacerbates the challenge. The scarcity of data could lead to overfitting, where the model learns the training data too well and performs poorly on unseen data. Therefore, it is imperative to employ strategies such as data augmentation or transfer learning to mitigate these issues and ensure robust and unbiased performance of the model.

Target	name	Count
Q14860489	Molecular function	110
Q16521	Taxon	52
Q2996394	Biological process	110
Q35120	Entity	55
Q5058355	Cellular component	110
Q7187	Gene	110
Q8054	Protein	110

Table 2.4: *BioTable CTA targets*

Furthermore, evaulate the model on a small dataset that is domain-specific can be promising for real-world applications as noted in the introduction data generation is one of the more expensive parts of any research task, and this task is not less any when training a machine

learning model, Veyhe et al. [36] showcased the challenges for domain-specific entity linking in tabular data based on syntax data soruce.

2.2.2 Exploration of Wikidata Subgraph Dataset

Wikidata, a comprehensive knowledge graph, comprises approximately 100 million entities. However, the BioTable dataset utilizes only a small subset of this knowledge graph. The Wikidata subgraph incorporated in the BioTable dataset is depicted in Table 2.4. To make the training feasible, a sub-knowledge graph was generated.

To generate a graph to train the model on, a subgraph was generted, and based on Section 1.3 and Hamilton et al. [6] that an neighbors sourouding the target node is needed therefor for traning the model, a subgraph was generted that with neighbors that are 2 hops from the target entities listed Table 2.4. To limited the types of nodes that are represented in the grpah the nodes are limtied to wikidata entities, meaning that the uuid for the entities starts with the letter Q.

Capturing all entities that are one or two hops away results in a subgraph with $62,175$ entities in the dataset with $1,019$ types of properties. To further limit the type of entities retrieved, the dataset was restricted to properties that link to entities internal to Wikidata. The selected properties were based on an internal Wikidata table showcasing all properties and their respective categories [37].

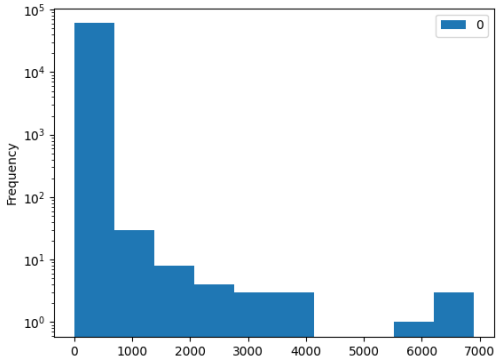


Figure 2.4: *Histogram of the number of neighbors for each node in the Wikidata knowledge graph*

The complexity and connectivity of an ontology can be inferred from the size of its neighbors. As depicted in Figure 2.4, the distribution of neighborhood sizes in the Wikidata subgraph is presented. A comparison of this distribution with that in Figure 2.2 reveals that the average neighborhood size of the Wikidata knowledge graph encompasses hundreds of entities. As discussed in Section 1.3, Graph Neural Networks (GNNs) consider the entire neighborhood when generating an embedding for a node. This process, however, results in substantial memory consumption. Further details on how this issue is addressed in terms of memory requirements are elaborated in Chapter 4.

2.2.3 Analysis of the WDC Schema.org Table Annotation Benchmark Dataset

The WDC Schema.org Table Annotation Benchmark (SOTAB) dataset is a comprehensive collection of 48,379 tables, with 46,790 tables allocated for training purposes. These tables span across 17 diverse domains, providing a broad spectrum of general domains for analysis (see Table 2.5).

Domain	**Train**	**Test**
Museum	116	14
MusicAlbum	316	38
TVEpisode	346	45
SportsEvent	613	86
Restaurant	1033	135
Hotel	1393	238
Movie	1846	202
Place	2015	224
Book	2022	341
MusicRecording	3344	401
CreativeWork	3393	686
JobPosting	3584	484
LocalBusiness	3744	602
Person	3818	374
Recipe	4083	412
Event	4721	593
Product	10403	875
Total	46790	5732

Table 2.5: *WDC Schema.org Table Annotation Benchmark dataset table per domain domains*

A closer examination of the training and validation datasets reveals a significant representation of entities. The training dataset comprises 130,472 entities, while the validation dataset contains 16,841 entities. Both datasets share 87 unique entities (see Table 2.6).

	Training	Validation
Entities	130,472	16,841
Unique entities	87	87

Table 2.6: *WDC Schema.org Table Annotation Benchmark dataset statistics*

However, an imbalance is observed in the frequency of entities in the training dataset, as depicted in Figure 2.5. The entity 'Name' is the most prevalent with a count of 18,033, followed by a steep decline in frequency for the subsequent entities. This class imbalance could potentially bias the training towards over-represented classes, as discussed in Section 1.3.

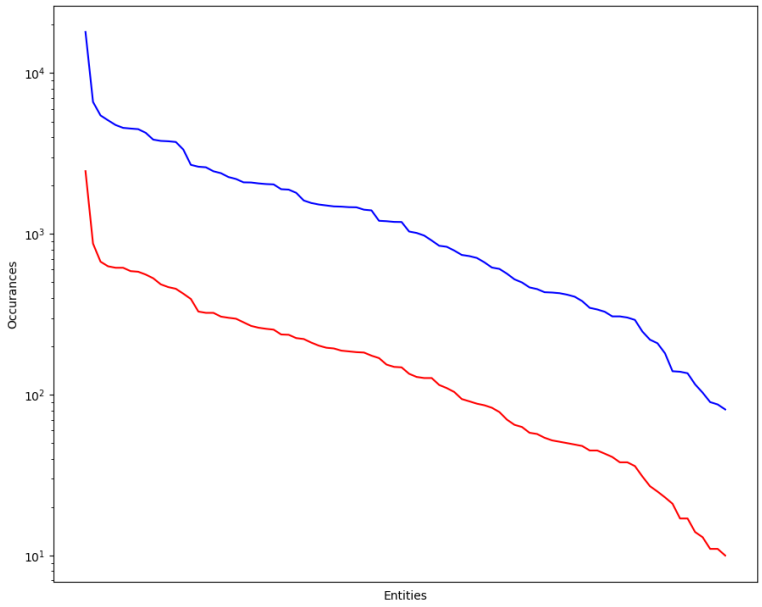


Figure 2.5: *Histogram of entity frequency in the WDC Schema.org Table Annotation Benchmark training dataset, with the y-axis on a log scale.*

2.2.4 Insights into the Schema.org Dataset

The Schema.org ontology, a collaborative initiative supported by major search engines like Google, Microsoft, Yahoo, and Yandex, aims to develop and promote schemas for structured data on the internet. The ontology serves as a common vocabulary for webmasters to structure their website data in a manner comprehensible to these search engines.

The ontology currently encompasses over 600 distinct types of entities, each with its own set of properties. These entities range from general concepts such as 'Thing' and 'Place' to more specific ones like 'Movie' and 'Person'. The dataset used for the table annotation task comprises approximately 130,000 entities, which are instances of the types defined in the Schema.org ontology. Each entity in the dataset possesses a set of properties filled with data (see Figure 2.5 for a breakdown of entity representation in the training dataset).

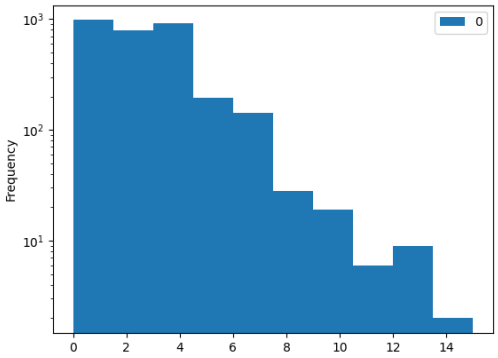


Figure 2.6: *Histogram of the number of neighbors for each node in the Schema.org knowledge graph*

The table annotation task, specifically the SOTAB task, is intrinsically linked to the Schema.org ontology. The task's objective is to annotate tables with entities and properties from the Schema.org ontology, which involves identifying the entities in the table, determining their types, and assigning properties to them. The Schema.org ontology provides the necessary vocabulary and structure for this task, enabling a coherent interpretation of the data in the tables (see Figure 2.6 for a histogram of the number of neighbors for each node in the Schema.org knowledge graph).

3 | Model architecture

This section provides an in-depth exploration of the methodologies employed in the project, focusing on the concepts of text embedding and mention embedding. The input data for the project is categorized into text data and tabular data, each with its unique input structure in the BERT tokenizer. The text data input structure is based on the methodology proposed by Wu et al. [2], where each mention in the corpus is encapsulated within a specific string format (Example 3.2.1). On the other hand, the tabular data input structure is inspired by the work of Trabelsi et al. [3], but with a simplified format due to the specific requirements and constraints of the project (Example 3.2.2).

Following the discussion on input data, the section delves into the architecture of the Mention Encoder, a model that leverages the BERT model [4] with an additional projection head. The Mention Encoder's architecture, including the tokenization process and the role of the projection head, is discussed in detail, providing a comprehensive understanding of the model's functionality (Figure 3.2). This chapter serves as a foundation for understanding the methodologies and techniques employed in the project, paving the way for subsequent discussions on the project's implementation and results.

3.1 Projection Head

The concept of a projection head, a fundamental component in machine learning, is utilized to transform input data, specifically embeddings, into a different space, thereby generating projected embeddings [38]. The projection is accomplished through a series of operations collectively known as projection layers. The projection head is a simple feed-forward neural network that serves to project the entity and mention embeddings to the same dimensionality [39]. It is essentially a fully connected layer equipped with a ReLU activation function. The output generated by the projection head comprises the entity and mention embeddings. The primary function of the projection head is to reduce the dimensionality of the embeddings, thereby making the embeddings comparable.

The initialization of the projection head involves several parameters: the number of input channels, the dimension of the projection, the number of projection layers, and the dropout rate. The dropout rate is a regularization technique employed to prevent overfitting in the model. It achieves this by randomly setting a fraction of input units to 0 at each update during the training phase.

The projection head comprises a list of projection layers. The first layer is a linear transformation of the input data. Subsequently, a GELU activation function is applied for each projection layer, followed by another linear transformation, a dropout operation, and finally, a layer normalization.

The GELU activation function introduces non-linearity into the model, thereby enabling it to learn more complex patterns. Layer normalization is a technique that normalizes the features of each individual sample independently of other samples.

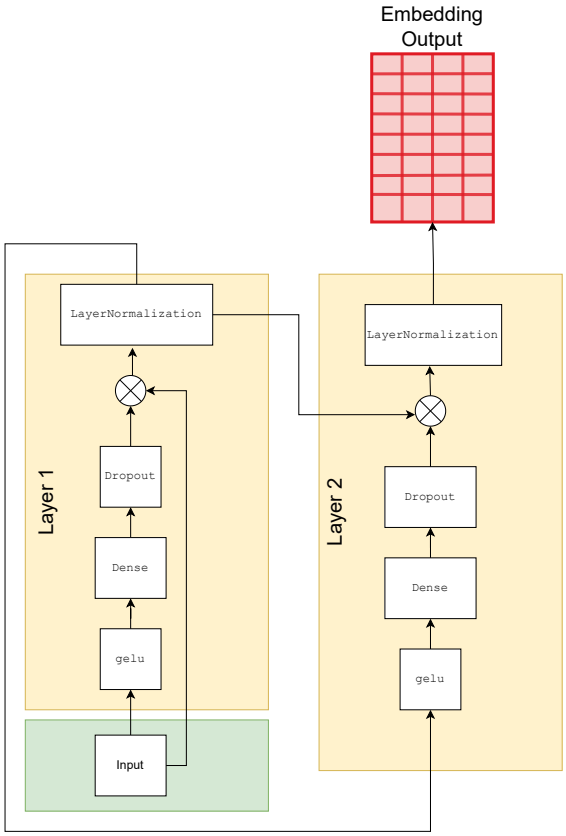


Figure 3.1: *Projection layer architecture*

The forward pass of the projection head takes the input embeddings and applies the projection layers to them sequentially. If the current layer is a linear layer, it applies the previous layer (GELU), the current layer (Linear), the next layer (Dropout), and then adds the result to the original input embeddings. This is followed by applying the layer normalization. The outcome of this process is the projected embeddings.

The projection head proves to be a valuable tool in numerous machine learning tasks, such as dimensionality reduction, feature extraction, and data visualization. It facilitates the transformation of input data into a different space, thereby enabling more complex patterns to be learned and utilized.

3.2 Mention encoder model

The Mention Encoder model is a sophisticated architecture that leverages the power of the BERT model with an additional projection head. The input data for this model is divided into two categories: text data and tabular data. For text data, each mention in the corpus is encapsulated within a string and structured as per the methodology proposed by Wu et al. For tabular data, the project aims to perform Column Type Annotation, with the input into the BERT tokenizer inspired by the work of Trabelsi et al. The Mention Encoder's architecture includes the BERT model and a projection head, which allows the model to project the output of the BERT model into a lower-dimensional space, enabling efficient computation and storage. The input to the Mention Encoder is tokenized text, a process that involves breaking down the text into individual words or tokens.

3.2.1 Input

The input data for this project is divided into two categories: text data and tabular data. For text data, the input into the BERT tokenizer is structured as per the methodology proposed by Wu et al. [2]. Each mention in the corpus is encapsulated within a string of the format "[CLS] ctext_l [M_s] mention [M_e] ctext_r [SEP]". Here, ctext_l and ctext_r represent the left and right context of the mention, respectively. The context windows are selected such that the total length of the input equals 512 characters. The tokens M_s and M_e indicate the start and end of the mentioned token, while [CLS] and [SEP] are special tokens indicating the start and end of the input. An example is show in Example 3.2.1.

Example 3.2.1. To understand the the encodeing process of the mention encoder for text data, we will use the following quote is from the BB dataset with the mentions in bold, the input to the BERT tokenizer is shown in Table 3.1.

> **B. linens** was only isolated from the **Casera, Gorgonzola and Scimudin cheese surfaces**, confirming the more recent information that this species is not the most important bacterium on **smear cheeses**.

Mention	Input
B. linens	[CLS] [M_s] B. linens [M_e] was only isolated from the ... [SEP]
Casera, Gorgonzola and Scimudin cheese surfaces	[CLS] was only isoloated from the [M_s] Casera, Gorgonzola and Scimudin cheese surfaces [M_e], confirming the most ... [SEP]
smear cheeses	[CLS] ... confirming the more recent information that this species is not the most important bacterium on [M_s] smear cheeses [M_e]. [SEP]

Table 3.1: *Input data for the Mention Encoder.*

◀

For tabular data, the project aims to perform *Column Type Annotation*. The input into the BERT tokenizer is inspired by the work of Trabelsi et al. [3] and is formatted as "[CLS] Query [SEP] Metadata [SEP] [header name] [type] [value] [SEP] [header name] [type] [value] ...". However, in this project, the input is simplified to "[CLS] Value_1 [SEP] value_2 [SEP] ...". This is because only the values of the cells in the tabular data are used, while the header names and types are ignored. This decision is based on the fact that the table may not always contain header information and type identification is beyond the scope of this project. Additionally, the provided dataset does not contain metadata or query information, hence they are also excluded from the encoding. An example is shown in Example 3.2.2.

Example 3.2.2. To understand the enocdeing for tabular data for the CTA task, Table 3.2 is from the WDC dataset that listed event that are scheduled.

2019-06-20T18:30-MDT	Why Developers Should Care About Service Mesh | EnvZone	2019-06-20T21:00	EventScheduled
2019-06-12T17:30-MDT	Dissolving The Problem: Kafka Is More ACID Than Your Database With Tim Berglund | EnvZone	2019-06-12T20:30	EventScheduled
	Lakewood Networking Referral Group | EnvZone		EventScheduled
	Mobile Apps For Photographers With Gary White | EnvZone		EventScheduled
	5 Tips To Make Your Website Awesome | EnvZone		EventScheduled
	Wheat Ridge Networking Referral Group | EnvZone		EventScheduled
	Highlands Ranch Business Networking Happy Hour | EnvZone		EventScheduled
2020-09-26T19:40-MDT	Moving To React And Debugging Live | EnvZone	2019-06-18T21:00	EventScheduled
	Machine Learning With Google Developer Expert Evan Hennis | EnvZone		EventScheduled

Table 3.2: *Tabular dataset from Event Zone.*

To shown the encodeing process the first coloum in Table 3.2 for the BERT tokenizer is as follows:

- *[CLS] 2019-06-20T18:30-MDT [SEP] 2019-06-12T17:30-MDT [SEP] [SEP] [SEP] [SEP] [SEP] [SEP] 2020-09-26T19-40-MDT [SEP].*

◀

3.2.2 Mention Encoder Architecture

The Mention Encoder, as proposed by Wu et al. [2], is a sophisticated model that leverages the power of the BERT model [4] with an additional projection head. The BERT model, a transformer-based machine learning technique for natural language processing, forms the base of the Mention Encoder. The projection head, which is a linear layer, is stacked on top of the BERT model. This architecture allows the model to project the output of the BERT model into a lower-dimensional space, thereby enabling efficient computation and storage. The specifics of the projection head are discussed in detail in the previous section of this thesis.

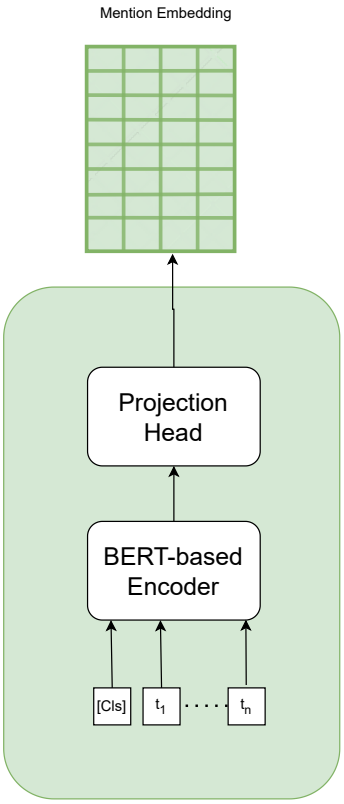


Figure 3.2: *The architecture of the Mention Encoder.*

The input to the Mention Encoder is tokenized text. The process of tokenization, which involves breaking down the text into individual words or tokens, is a crucial step in natural language processing. The tokenization process for the Mention Encoder is explained in detail in the above subsection.

The architecture of the Mention Encoder is illustrated in Figure 3.2. This figure provides a visual representation of the model, including the BERT model and the projection head, and how they interact to process the tokenized input.

3.3 Entity Embedding model architecture

This section provides an in-depth exploration of the Entity Embedding model architecture, which is a crucial component of our research. The model architecture is divided into two main subsections: Input Encoding and Ontology Embedding Model. The former discusses the input format for the ontology embedding process, which is derived from various ontologies/knowledge graphs, as detailed in Section 2. The latter delves into the model architecture for ontology embedding, which is based on the work of Wu et al. [2] and Louis et al. [24]. The model's flexibility, particularly in the utilization of Graph Neural Network (GNN) layers, is a key feature that allows it to be tailored to specific requirements and scenarios. This section also presents a comprehensive comparison of different GNN layers and their edge attributes. The final part of this section discusses the projection head, which is a linear layer that projects the output of the BERT model into a lower-dimensional space. The specifics of the projection head are discussed in detail in Section 3.1. This section serves as a foundation for understanding the subsequent analysis and results of our

research.

3.3.1 Input encoding

The input for the ontology embedding process is derived from the ontologies/knowledge graphs discussed in Section 2. Each entity in these ontologies contains a name, and a subset of them also includes one or more of the following: a description, synonyms, or comments. As noted in Section chapter 2, all ontologies except the OBO knowledge graph have multiple relationship types. However, for simplicity, edge attributes are not considered in this project, leaving room for future work [40, 41]. The entity input format follows the approach of Wu et al. [2], which is "[CLS] title [ENT] description [SEP]". The knowledge bases Wikidata, Schema.org, and OntoBiotope, presented in Section 2, provide the appropriate stand-ins for *title* and *description*.

3.3.2 Ontology Embedding model

The model architecture for ontology embedding is based on the work of Wu et al. [2] and Louis et al. [24]. The entity encoder, shown in Figure 3.4a, is similar to the one from Wu et al. [2], but it includes a projection layer on top. The model by Louis et al. [24], shown in Figure 3.3, uses a BERT-based encoder for article embedding and enriches the embedding with structural information using a Graph Neural Network (GNN). In contrast, our proposed model architecture, shown in Figure 3.4b, integrates these steps.

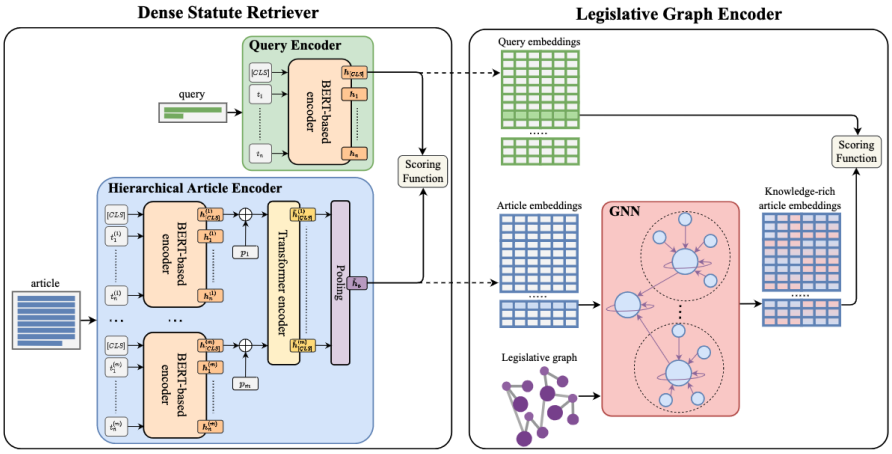


Figure 3.3: *The GDSR model architecture [24].*

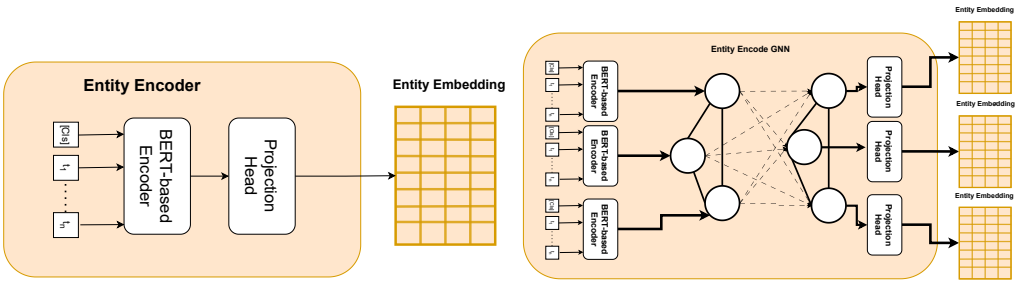
The flexibility of the model layer in our proposed system is one of its key features, as it allows for the utilization of any of the Graph Neural Network (GNN) layers, as detailed in Table 3.3. This adaptability is crucial in the context of our research, as it enables the model to be tailored to specific requirements and scenarios. The ontologies' features, as elaborated in chapter 2, are primarily concentrated on node features, rather than edge features. This distinction is not trivial, as it significantly influences the functionality and performance of our model. To assess the effectiveness of enriching the embedding with a GNN layer, we have designed the model to be configurable, allowing for the inclusion or exclusion of the GNN layers. This design choice provides a robust framework for evaluating the impact of GNN layers on the overall performance of the model. The two possible configurations, which represent different approaches to integrating GNN layers, are illustrated in Figure 3.4.

This comprehensive overview of the model layer and its configuration options provides a solid foundation for understanding the subsequent analysis and results of our research.

Method	Citation	Edge attributes
Graph Convolutional Networks (GCN)	Kipf et al. [12]	✗
GraphSAGE	Hamilton et al. [6]	✓
Graph Attention Networks (GAT)	Velickovic et al. [42]	✓
k-dimensional GNNs (K-GNN)	Morris et al. [43]	✓
Graph Attention Networks Verson 2 (GATv2)	Brody et al. [44]	✓

Table 3.3: *Comparison of different Graph Neural Network layers*

The projection head, discussed in section 3.1, is the same as the one used in the Mention Encoder. The final model architecture, which combines the two models, is presented in section 3.4. The projection head is a linear layer that projects the output of the BERT model into a lower-dimensional space. The specifics of the projection head are discussed in detail in Section 3.1.



(a) Entity Encoder model architecture *(b) Entity Encoder model architecture with GNN*

Figure 3.4: *Entity Encoder model architectures*

3.4 Dual Encoding Model Architecture

This section provides a comprehensive overview of the dual encoding model architecture. As underscored by Dong et al. [38], there exists a variety of dual encoder architectures, including but not limited to Siamese Dual Encoder (SDE), Asymmetric Dual Encoder (ADE), ADE with Shared Token Embedding (ADE-STE), ADE with Frozen Token Embedding (ADE-FTE), and ADE with a yet to be defined component (ADE-SPL). The primary focus of this project is the ADE model architecture, which is further bifurcated into two main components: the entity encoder and the mention encoder, as elaborated in sections 3.3 and 3.2, respectively. This architecture facilitates simpler modifications to the different components of the dual encoder, thereby enabling each stack to adapt and better fit the conclusion. The selection of the loss/score function, a critical aspect of the model, will be discussed in the subsequent section.

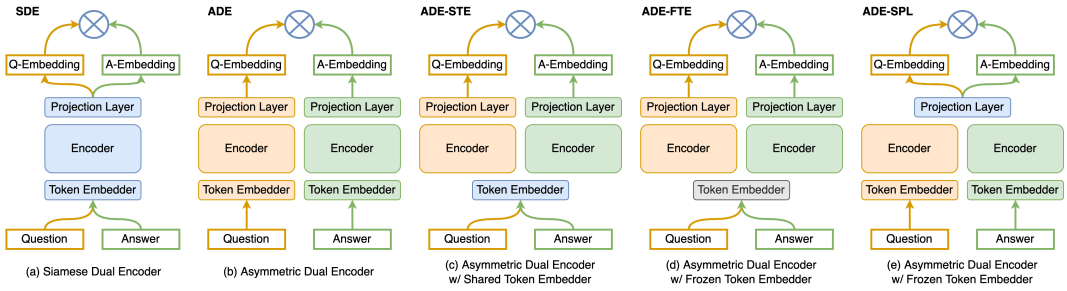
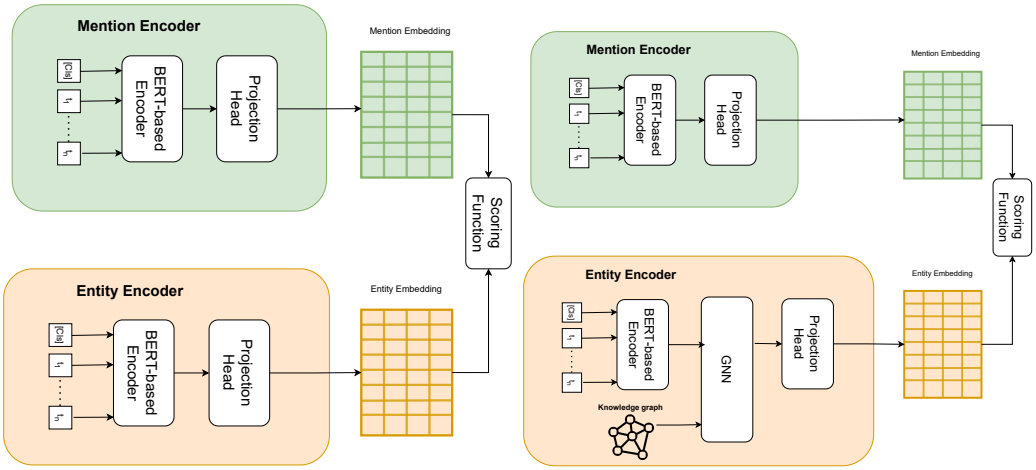


Figure 3.5: *Architectures of dual encoders [38].*

As previously discussed in Section 3.2 embedding, the model architecture remains static, with the only modification being the number of projection layers incorporated. This is a crucial aspect of the model's design, as it allows for flexibility in the model's complexity without necessitating a complete overhaul of the architecture. In contrast, as outlined in Section 3.3, the model architecture can adopt a bi-directional encoding structure based on BERT if the Graph Neural Network (GNN) model is omitted. In this scenario, both encoders will resemble each other, with the only difference being the number of input structures, resulting in the dual encoder architecture as shown in Figure 3.6a. However, the inclusion of the GNN layer introduces a new dimension to the model architecture, as depicted in Figure 3.6b. The GNN layer allows the model to capture and process more complex relationships between entities, thereby enhancing the model's performance in tasks requiring a deeper understanding of the data's structure.



(a) *Dual encoding model architecture.* ***(b)*** *Dual encoding model architecture with GNN.*

Figure 3.6: *Dual encoding model architectures.*

3.5 Scoring function

This thesis delves into the critical role of the loss function in steering both encoders to acquire identical representations. The score function must be adept at evaluating the similarity between mention and entity embeddings. Furthermore, it is essential for practical applications that the entity's embedding can be computed offline. The inference should be achievable by calculating the mention embedding and retrieving the nearest k neighbors in less than a minute, even in extensive knowledge bases like DBpedia, which encompasses

billions of entities.

This study presents three subsections, each focusing on a different function relevant to the thesis. The first function, Cosine similarity/dot function, introduces the cosine embedding loss. This is a common scoring function that enables dense retrieval based on the angular similarity of embeddings representing entities. The second function, Triplet margin loss, introduces the triplet loss function. This is a common loss function that allows for dense retrieval based on the Euclidean similarity of embeddings representing entities. From the perspective that embeddings are maps from higher dimensionality into a manifold in lower dimension, the idea behind triplet loss is to move similar entities closer together, and dissimilar entities farther away. The third function, with slight abuse of notation cross-entropy, is a scoring function aimed at classification. However, the aim of the function is to maximize the value of the right "class" and minimize the value to the other negative anchors. This can be seen as an updated triplet margin loss where the positive anchor is pushed closer and the remainder is pushed farther away. These three functions are presented because they provide a comprehensive understanding of the scoring and loss functions used in machine learning, which is crucial to the thesis.

3.5.1 Cosine Embedding Loss as the similarity Function

This study employs the cosine embedding loss as the scoring function. The cosine embedding loss quantifies the dissimilarity between two non-zero vectors within an inner product space by measuring the cosine of the angle between them. This dissimilarity is computed as the dot product of the normalized vectors, as defined in Eq. 3.1:

$$\text{Cosine embedding loss} = L_c(\mathbf{A}, \mathbf{B}) := 1 - \cos(\theta) = 1 - \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}. \tag{3.1}$$

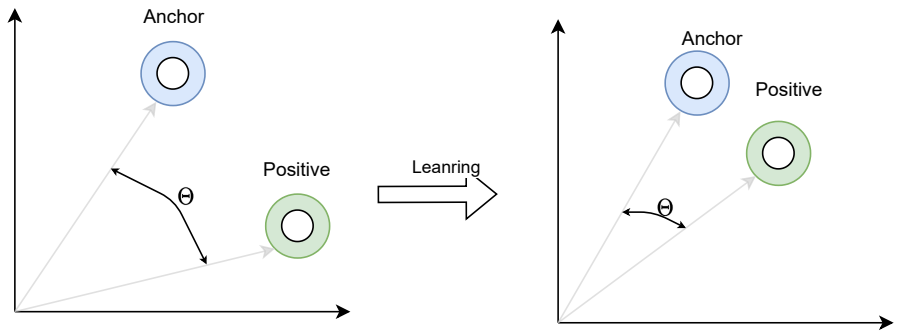


Figure 3.7: *Cosine Similarity*

While cosine similarity is an effective measure of similarity between two vectors, it is important to note that if KG is a given knowledge graph with V representing all entities in the knowledge base, then for each $\mathbf{v_i} \in V$ for $i \in \{1, 2, \ldots, \mid V \mid\}$, $B_i = \text{enc}_e(v_i)$ represents the entity encoding. Let m be a mention in a text, then $\mathbf{A} = \text{enc}_m(m)$ represents the mention encoding. To find the k closest $\mathbf{B}_j$, it is necessary to iterate over all $\mathbf{B_i}$ and calculate the cosine similarity between $\mathbf{A}$ and $\mathbf{B_i}$. Consequently, the operation becomes $\mathcal{O}(\mid V \mid)$. Johnson et al. (2019) [45] introduced GPU-based similarity search that can scale to billions of similarity searches and proposed an approximate algorithm.

3.5.2 Cross Entropy Loss as the Loss Function

In the field of machine learning, particularly in classification tasks, the Cross-Entropy Loss function plays a pivotal role. It is defined as the negative log-likelihood of the correct class given the input, making it a popular choice due to its differentiability and ease of optimization [7].

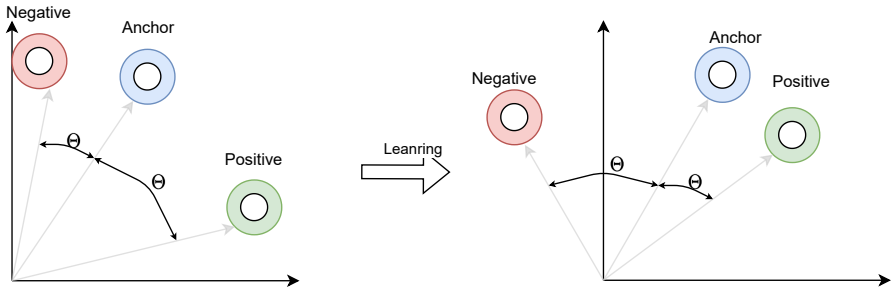


Figure 3.8: *Cross Entropy Loss*

In the context of vector embeddings, an important operation is the concatenation of Positive and Negative vector embeddings. This results in a vector, denoted as C, which is expressed as $C = [p_1, p_2, \ldots, p_k, n_1, n_2, \ldots, n_k]$. The vector C belongs to the set of real numbers and has dimensions $2k \times embdim$. Concurrently, another vector, E, is defined as $E = [e_1, e_2, \ldots, e_k]$.

The mathematical relationship between these vectors is established through the dot product of E and the transpose of C, resulting in a new vector, S. This relationship is represented as $E \cdot C^T = S$. Similar to C, the vector S also belongs to the set of real numbers and has dimensions $k \times 2k$.

In terms of loss functions, the built-in cross-entropy loss function in Python, 'torch.nn.function.cross_entropy', is utilized. This function's target is $[0, 1, \ldots, k]$. This configuration informs the loss function that for row i, the similarity for column i should be maximized. This corresponds to the positive item, and the rest should be as close to zero as possible [24].

3.5.3 Triplet Loss and Euclidean Similarity Search

The following paragraph is a comprehensive overview of the application of the Triplet Loss Function in computer science, particularly in tasks related to computer vision and similarity search.

The Triplet Loss Function, originating from the field of machine learning, is a powerful tool frequently employed in tasks related to computer vision, such as image recognition [46]. This function operates on a triplet of images: an anchor, a positive of the same class as the anchor, and a negative of a different class. The objective of the function is to ensure that the anchor is closer to the positive than to the negative by at least a certain margin, as mathematically formulated in Equation (1).

$$L(A, P, N) = max(||f(A) - f(P)|| - ||f(A) - f(N)|| + margin, 0) \tag{3.2}$$

In this equation, A represents the anchor, P the positive, N the negative, f the embedding function that maps an image to a vector, and $||.||$ denotes a distance metric, specifically the L_2 norm in this project.

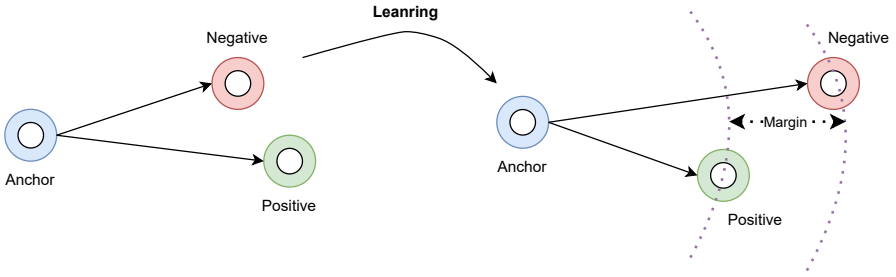


Figure 3.9: *Triplet Loss between Anchor, Positive and Negative*

The Triplet Loss Function has proven effective in tasks such as face recognition, where distinguishing between different individuals is crucial [46]. However, the selection of the triplets requires careful consideration to ensure the effectiveness of the training process. Louis et al. [24] demonstrated the applicability of the triplet loss for dense retrieval in combination with Graph Neural Networks (GNNs), albeit without utilizing a dual encoder architecture.

$$\text{Euclidean similarity} = L_e(\mathbf{A}, \mathbf{B}) := \frac{1}{1 + \|\mathbf{A} - \mathbf{B}\|_2}. \tag{3.3}$$

In the context of similarity search Equation 3.3, the triplet loss function facilitates a Euclidean similarity search, which is significantly faster than a cosine similarity search. This speed-up can be crucial in large-scale applications where efficiency is a key concern. Furthermore, the use of indexing for Euclidean-based distance, a well-understood concept from database theory, allows for faster retrieval of data, further enhancing the efficiency of the similarity search process [47].

4 | Experiment setup

In this chapter, we delve into the experimental setup for validating our model, focusing on the assumptions, data loading configurations, and model configurations. We begin by establishing the assumptions concerning the dataset we are working with. We adopt the closed-world assumption for the knowledge graph, implying that all relevant concepts are included in the graph [5]. We also assume that the Graph Neural Network (GNN) model is transductive and that we have pre-computed all entity embeddings in the knowledge graph offline at the time of inference.

In the data loading configurations, we address the issue of class imbalance in the dataset, implementing an undersampling approach based on the methodology proposed by Hamilton et al. [6]. We describe the process of dividing the mentions into batches, generating a subgraph for each batch, and selecting positive and negative mentions for each entity.

In the model configurations, we elucidate the configurations of the models utilized in the experiments, which are summarized in Table 4.2. We highlight the challenges posed by the stringent time constraint of 4 hours for model training and the implications of this limitation on the interpretation of the results. We also provide a comparison with the work of Wu et al. [2] and detail the hardware utilized for training the models.

4.1 Validation Assumptions

Before proceeding with the validation of our model, it is essential to establish certain assumptions concerning the dataset we are working with. First, we adopt the closed-world assumption for the knowledge graph, which implies that if a concept is relevant to the domain, it is included in the knowledge graph [5]. Consequently, any concept not present in the knowledge graph is considered false. Furthermore, as every concept relevant to the domain is in the knowledge graph, we can also conclude that every mention in the corpus corresponds to a concept in the knowledge graph. Therefore, we do not encounter false mentions during training and inference, and there are no true negative samples in the inference step.

Second, we assume that the Graph Neural Network (GNN) model is transductive. This means that the model can only make inferences on the entities present in the knowledge graph and cannot infer entities not included in the knowledge graph.

Third, we assume that we have pre-computed all entity embeddings in the knowledge graph offline at the time of inference. The inference step is essentially an entity suggestion task. If the true entity is among the top k results, it is considered a predicted positive k (PP_k) case. If not, it is considered a predicted negative k (PN_k) case. As per the second assumption, each mention has a corresponding entity at the time of inference, and all mentions are true conditions. Therefore, we can only measure True Positive (TP) and False Negative (FN) cases, limiting the standard performance measurement to recall. In this project, $recall_k$ implies that the correct entity was among the top k entities encoding most similar to the

given mention's encoding, as summery of the apprivations are listed in Table 4.1.

Symbol	Description
TP	True Positive
FN	False Negative
$recall_k$	Recall at k
PP_k	Predicted Positive at k
PN_k	Predicted Negative at k

Table 4.1: *Symbols and their descriptions*

4.2 Dataloading configurations

Chapter 2 underscores the prevalent issue of class imbalance, as depicted in figures 2.1, 2.5, and Table 2.4. In the Biotable dataset, the Taxon and Entity entities are underrepresented, creating a skewed distribution of classes. Similarly, the WDC dataset reveals a stark disparity in the frequency of certain entities, with some appearing up to 100 times more frequently than others. This imbalance poses significant challenges in the training and evaluation of Neural Networks, as discussed in Section 1.3. To address this issue, we have implemented an undersampling approach. This strategy involves designating the primary dataset as the knowledge graph and structuring the data loading process based on the methodology proposed by Hamilton et al. [6]. This approach aims to ensure a more balanced representation of classes, thereby enhancing the performance and reliability of the Neural Networks.

To initiate the sampling process, we first assume that $M = \{(m_i, v_i)\}$ represents the mention dataset, where m_i is the feature vector and v_i is the target entity. We also assume that $KG = (V, E, F)$ represents the knowledge graph, and that $\bigcup_{(m,v)\in M} \subseteq V$. We compiled a list of entities in the mention dataset for data sampling. Let $V_m = \bigcup_{(m,v)\in M}$ denote a set of entities in the mention dataset, and let n represent the desired batch size. The mentions in V_m were divided into batches, each containing n entities. A subgraph was generated for each batch using the entities in the batch as the seed. The neighbors for each entity were then loaded. The neighborhood graph is defined as $\bigcup_{v\in V_i} \mathrm{N}_k(v)$, and $E_s = \{(v_i, v_j) \mid v_i, v_j \in V_s \wedge (v_i, v_j) \in E\}$, where $\mathrm{N}_k(v)$ loads k arbitrary neighborhood vertices to v. Each batch is denoted as V_i.

For each entity $v_i \in V_i$, the positive and negative mentions are selected as follows: $p_i \in \{(m_j, v_j) \mid v_j = v_i\}$ and $n_i \in \{(m_j, v_j) \mid v_j \neq e_i\}$, where p, n are chosen arbitrarily. This leads to the final format of the input data being defined as $d_i = (v_i, p_i, n_i)$.

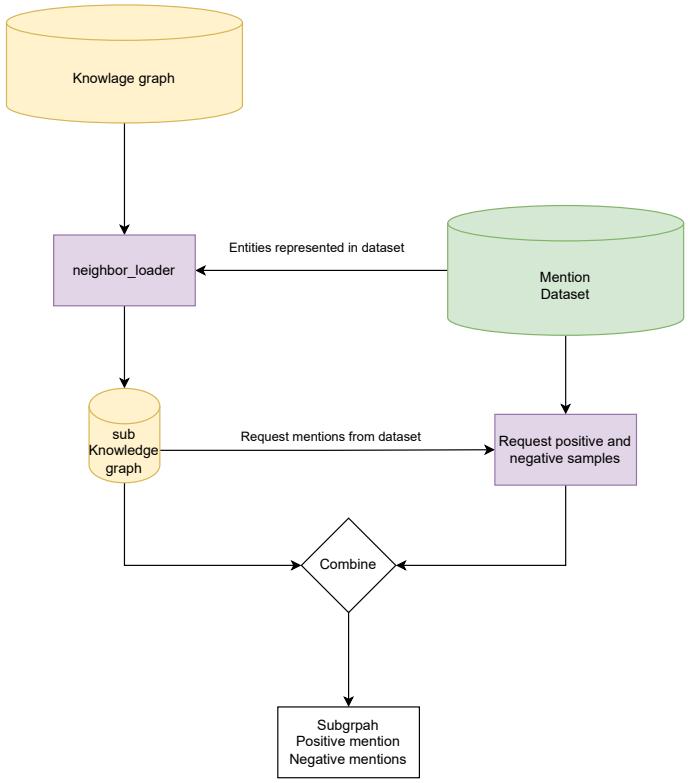


Figure 4.1: *Data loading flow*

4.3 Model Configurations

This section elucidates the configurations of the models utilized in the experiments, which are comprehensively summarized in Table 4.2. The models undergo training within a stringent time constraint of 4 hours. This time limit is imposed to ensure an equitable learning environment for all models. However, this time constraint also presents a significant challenge. The extensive duration required to train each model, coupled with the 4-hour time limit, allows for each configuration to be executed only once. This limitation precludes the possibility of performing a statistical analysis of the results, which could potentially provide more robust and reliable insights. Therefore, the results presented in this study should be interpreted with this limitation in mind. Future work could potentially involve extending the time limit or employing more efficient training methods to allow for multiple executions of each configuration, thereby enabling a more comprehensive statistical analysis.

The entity encoder models in Table 4.2 with zero GNN layer count operated in the same manner as a bi-directional bert-based encoder for entity encoding. The cosine similarity is equivalent to the dot similarity if the vectors are normalized, providing a comparison with Wu et al. [2].

Table 4.2 presents the configurations of the models used in the study. The Bert Models used are 'base-uncased' and 'tiny', both pre-trained models with weights provided by Hugging Face. The loss functions used are Cross-Entropy loss with cosine similarity (CELCS) and Triplet Margin Loss (TML). The datasets used are BB, BioTable, and WDC. The GNN layers count varies from 0, 3, to 5.

The hardware utilized for training the models comprised of 4x Nvidia L4 GPU graphic cards, each with 24GB of memory. The batch size for each model was kept constant

throughout the training process. This resulted in a total of 36 runs, or equivalently, 144 hours of training time to train all the models. The results of these training sessions are presented in the subsequent chapter.

In the course of model training, a time constraint of 4 hours was enforced to ensure equitable learning. Other potential methods to maintain consistency include keeping the number of steps or epochs constant. However, each configuration was executed only once, which precludes the possibility of performing a statistical analysis of the results. This limitation is due to the extensive time required to train each model and the 4-hour time limit. To make a comparison with Wu et al. [2], the entity encoder models in Table 4.2 with zero GNN layer count operated in the same manner as a bi-directional bert-based encoder for entity encoding, and the cosine similarity is equivalent to the dot similarity if the vectors are normalized.

Bert Model	**Loss function**	**Dataset**	**GNNLC**
base-uncased	CELCS	BB	0
tiny	CELCS	BB	0
base-uncased	TML	BB	0
tiny	TML	BB	0
base-uncased	CELCS	BioTable	0
tiny	CELCS	BioTable	0
base-uncased	TML	BioTable	0
tiny	TML	BioTable	0
base-uncased	CELCS	WDC	0
tiny	CELCS	WDC	0
base-uncased	TML	WDC	0
tiny	TML	WDC	0
base-uncased	CELCS	BB	3
tiny	CELCS	BB	3
base-uncased	TML	BB	3
tiny	TML	BB	3
base-uncased	CELCS	BioTable	3
tiny	CELCS	BioTable	3
base-uncased	TML	BioTable	3
tiny	TML	BioTable	3
base-uncased	CELCS	WDC	3
tiny	CELCS	WDC	3
base-uncased	TML	WDC	3
tiny	TML	WDC	3
base-uncased	CELCS	BB	5
tiny	CELCS	BB	5
base-uncased	TML	BB	5
tiny	TML	BB	5
base-uncased	CELCS	BioTable	5
tiny	CELCS	BioTable	5
base-uncased	TML	BioTable	5
tiny	TML	BioTable	5
base-uncased	CELCS	WDC	5
tiny	CELCS	WDC	5
base-uncased	TML	WDC	5
tiny	TML	WDC	5

Table 4.2: *Table 4.2 presents the configurations of the models used in the study. The Bert Models used are 'base-uncased' and 'tiny', both pre-trained models with weights provided by Hugging Face. The loss functions used are Cross-Entropy loss with cosine similarity (CELCS) and Triplet Margin Loss (TML). The datasets used are BB, BioTable, and WDC. The GNN layers count (GNNLC) varies from 0, 3, to 5.*

The hardware utilized for training the models comprised of 4x Nvidia L4 GPU graphic cards, each with 24GB of memory. The batch size for each model was kept constant throughout the training process. This resulted in a total of 36 runs, or equivalently, 144 hours of training time to train all the models. The results of these training sessions are presented in the subsequent chapter.

5 | Evaluation of Experiments

The experimental setup for this study is detailed in Table 4.2. The subsequent sections provide an in-depth analysis of the results derived from these experiments, with a particular emphasis on the influence of the configuration on the final outcomes. Due to constraints in resources, each configuration is executed only once. For a more robust statistical understanding of the significance of each configuration, it would be necessary to conduct additional runs. However, due to time limitations, not all configurations of the model with the text encoder being bert-base-uncased were executed, with 4 runs remaining incomplete. The results of the executed runs are presented in Table 5.1.

Table 4.2 offers a comprehensive overview of all the model configurations that were evaluated for performance. However, as indicated in Table 5.1, not all configurations were executed. Certain configurations, specifically those utilizing bert-base-uncased and the triplet margin loss function for the Schema and BioTable datasets, were not executed due to time constraints.

The model architecture, which includes a Bi-directional encoder as the entity encoder and employs cosine similarity measurement, is based on the work of Wu et al. [2]. This model currently represents the state-of-the-art in the field of dense retrieval and entity suggestion with a knowledge graph.

Table 5.1 lists all model configurations that were trained in this project. The highest $recall_k$ for $k \in \{10, 50, 100\}$ measured for any run, and the minimum of the loss score found, are discussed in detail later in this chapter.

Bert encoder	GNN	Dataset	Loss function	Recall 10	Recall 50	Recall 100	Loss Score
tiny	0	BB	CELCS	8.04	21.34	29.38	1.68
tiny	3	BB	CELCS	10.54	19.46	26.88	1.68
tiny	5	BB	CELCS	8.48	19.91	27.05	1.70
tiny	0	BioTable	CELCS	37.98	37.98	52.26	0.46
tiny	3	BioTable	CELCS	46.31	72.50	100.00	0.97
tiny	5	BioTable	CELCS	15.83	29.29	54.64	1.01
tiny	0	WDC	CELCS	15.21	41.40	60.14	2.02
tiny	3	WDC	CELCS	24.45	46.98	56.98	2.20
tiny	5	WDC	CELCS	23.57	51.26	63.00	2.15
tiny	0	BB	TML	4.22	7.03	10.27	0.66
tiny	3	BB	TML	2.46	6.61	13.08	0.59
tiny	5	BB	TML	3.68	8.24	13.75	0.67
tiny	0	BioTable	TML	41.34	44.11	100.00	0.31
tiny	3	BioTable	TML	23.42	42.53	100.00	0.45
tiny	5	BioTable	TML	42.50	70.30	100.00	0.37
tiny	0	WDC	TML	8.68	18.29	21.62	0.72
tiny	3	WDC	TML	9.21	27.72	38.13	0.71
tiny	5	WDC	TML	19.30	35.75	42.18	0.74
base-uncased	0	BB	CELCS	5.43	13.12	20.81	0.04
base-uncased	3	BB	CELCS	3.17	9.95	22.17	0.15
base-uncased	5	BB	CELCS	6.79	18.10	28.05	0.08
base-uncased	0	BioTable	CELCS	0.61	4.85	8.48	0.14
base-uncased	3	BioTable	CELCS	1.82	6.06	8.48	0.16
base-uncased	5	BioTable	CELCS	0.00	1.21	6.06	0.10
base-uncased	0	WDC	CELCS	8.43	19.87	31.92	0.14
base-uncased	3	WDC	CELCS	5.01	25.91	49.33	0.13
base-uncased	5	WDC	CELCS	16.68	47.20	59.06	0.16
base-uncased	5	WDC	CELCS	12.73	42.68	58.75	0.11
base-uncased	0	BB	TML	5.43	8.14	14.93	0.00
base-uncased	3	BB	TML	4.07	9.50	16.74	0.00
base-uncased	5	BB	TML	3.62	10.41	14.48	0.00
base-uncased	0	WDC	TML	2.15	9.46	19.63	0.00

Table 5.1: *This table presents the results of model experiments using different configurations of Bert encoders, GNNs, datasets, and loss functions. The Bert encoders used are bert-tiny and bert-base-uncased from huggingface. The loss functions include Cross-entropy loss with Cosine similarity (CELCS) and Triplet margin loss (TML).*

The first environment examined involves small bert models and the effect of the loss function on model validation. The mean results grouped by the loss function values are shown in Table 5.2. In all three metrics, the cosine cross entropy loss function performs better across the selected $Recall_k$. It is hypothesized that utilizing the cross entropy loss with cosine similarity improves performance. Note that the table only uses bert-tiny as most of the bert-base-uncased runs are for cosine similarity and not triplet loss.

	Recall 10	Recall 50	Recall 100
Cross entity loss with Cosine similarity	21.16	37.79	52.22
Triplet margin loss	17.20	28.95	48.78

Table 5.2: *Comparison of the mean performance of the cosine cross entropy loss function and the triplet margin loss function across different Recall values, using the bert-tiny model and the entire dataset.*

The next configuration examined is for entity encoding. The comparison is made if the encoder is a Bi-directional encoder, or if integrating the GNN layer into the entity encoder, as seen in Figure 3.4b, and changing the layers to three layers or five layers of GCNConv layers improves performance. Table 5.3 highlights the results from taking the mean from Table 5.1 when grouping by GNN layers. While for $Recall_{10}$ shows minimal difference of a single point,

however, the $recall_{50}, recall_{100}$ perform better when including the GNN entity in the model.

GNN layers	Recall 10	Recall 50	Recall 100
Bi-directional encoder	19.25	28.36	45.61
3	19.40	35.97	55.85
5	18.89	35.79	50.10

Table 5.3: *Comparison of recall values for different GNN layer configurations in the entity encoder model.*

Examining the effect of the dataset on the model, Table 5.4 shows the mean of the results from Table 5.1 when grouping by the dataset. The results show that the DBpedia dataset performs better than the other two datasets. This is due to the fact that the DBpedia dataset has more data than the other two datasets, and the model is able to learn more from the DBpedia dataset.

Dataset	Recall 10	Recall 50	Recall 100
BB	6.24	13.77	20.07
BioTable	34.56	49.45	84.48
WDC	16.74	36.90	47.01

Table 5.4: *Comparison of model performance across different datasets. The table shows the recall at 10, 50, and 100 for each dataset. It is evident from the results that the DBpedia dataset outperforms the other two datasets.*

The performance of the model on the DBpedia dataset surpasses that on the other two datasets, as seen in Table 5.4. This superior performance can be attributed to the size of the DBpedia dataset, which is the largest among the three. The model, therefore, has more data to learn from in the DBpedia dataset. However, a comprehensive understanding of the model's performance cannot be achieved by merely examining the overall results. It is essential to delve deeper into the results for each dataset and scrutinize the impact of the configuration on these results.

In Table 5.1, it is observed that some configurations with the bert-base-uncased model were not executed. To maintain fairness in comparisons, two separate tables are introduced. The first table includes only the matching configurations in bert-tiny, while the second table encompasses all runs from bert-tiny. This approach ensures that the comparisons are not skewed by the configurations that were not run.

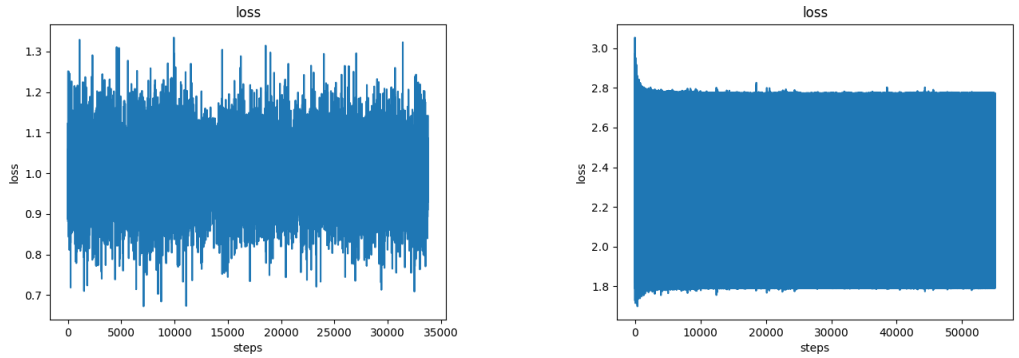
Table 5.1 and noting that some configurations with the bert-base-uncased model were not run, the configurations that were not run, and to keep the comparisons fair two tables are introduced one with only matching configurations in bert-tiny and one will all runs from bert-tiny.

	Recall 10	Recall 50	Recall 100
BERT	19.80	34.26	52.22
BERT Large	5.42	16.18	25.64

Table 5.5: *Comparison of Recall metrics between BERT and BERT Large models.*

Comparing the cross entropy loss model runs where the change is the BERT-uncased-model as seen in Table 5.5, it is seen that using the large bert model hinders the model's performance in all $recall_k$ metrics by a significant amount.

Figures 5.1a and 5.1b shows the loss over steps for the configuration bert=bert-tiny, gnn_layers=5, loss=triplet margin loss and cross entropy loss and dataset=BB. The loss function appears to be a random process with a constant mean, indicating that it does not decrease over time by any significant amount.

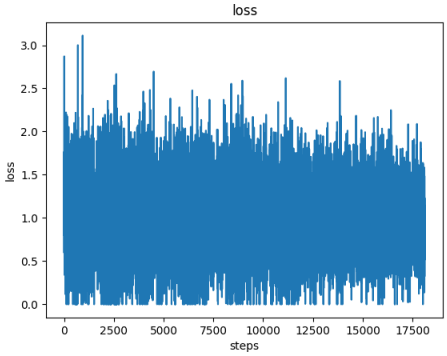


(a) *Loss over steps for the configuration bert=bert-tiny, gnn_layers=5, loss=triplet loss and dataset=WDC.*

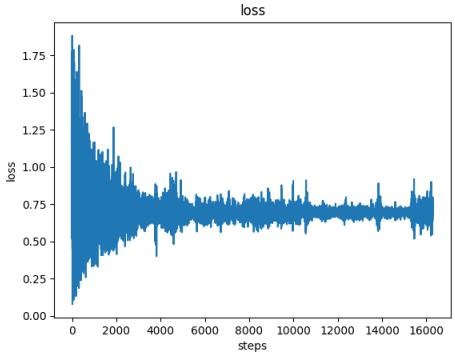
(b) *Loss over steps for the configuration bert=bert-tiny, gnn_layers=5, loss=cross entropy loss and dataset=WDC.*

Figure 5.1: *Illustration of loss over steps for the configurations bert=bert-tiny, gnn_layers=5, with two different loss functions: triplet margin loss and cross entropy loss, applied on the BB dataset. The loss function seems to follow a random process with a stable mean, suggesting no significant decrease over time.*

Figure 5.2a shows the loss over steps for the configuration bert=bert-base-uncased, gnn_layers=5, loss=triplet loss and dataset=BB. The loss function appears to be a random process with a constant mean, indicating that it does not decrease over time by any significant amount. However, Figure 5.2b, which shows the same configuration with the only change being the loss function changed to cross entropy loss, shows a decrease over time closing in on zero.



(a) Graphical representation of loss over steps for the configuration bert=bert-base-uncased, gnn_layers=5, loss=triplet loss and dataset=BB.



(b) Graphical representation of loss over steps for the configuration bert=bert-base-uncased, gnn_layers=5, loss=cross entropy loss and dataset=BB.

Figure 5.2: *Comparative analysis of loss over steps for different configurations.*

The loss over time with the change of bert-tiny and bert-base-uncased shows that the loss function for bert-base-uncased is lower. This is corroborated by Table 5.1 that shows the minimal loss recorded at any training point for bert-base-uncased models are lower by a large margin. This can indicating that for BERT-tiny that the triplet margin loss with random negative samples did not move the embeddings in the right direction, Schroff et al [46] noted that when introducing triplet margin loss that the mining strategy for the negative samples was importance for improving the models performances, and that it was importance to select hard negative samples.

In conclusion, it is found that using cross entity loss with cosine similarity performs better than triplet loss. Adding in the GNN into the entity encoder performs no difference when using recall 10, but performs better when using recall 50 and recall 100. Looking at the performances show that the model performs better on tabular dataset than on text based dataset. The model performance better on the Biotable dataset than WDC this can indicate that the model can performance well on domain specific datasets.

6 | Future Work

The successful implementation of the Graph Neural Network (GNN) layer in this project has demonstrated its potential in enhancing model performance. However, there are several areas that still require improvement.

Entity Embedding

The entity embedding utilized in this project was the small BERT model. However, other models, such as the RoBERTa model [48], which is larger than the BERT model, could potentially enhance the entity embedding and consequently improve the model's performance. The incorporation of the 'nil' entity, as suggested by [49], for entities not represented in the knowledge graph, could shift the knowledge graph assumption to the open world assumption. Building on the work of Louis et al. [24], the dense retrieval methodology could be used to fine-tune the bi-directional mention and entity encoder based on Wu et al. [2], and then use the GNN to enrich the embedding with graphical structural information.

GNN Layer

The GNN layer used in this project was the GAT layer. However, other GNN layers, such as the GraphSAGE layer [6], could potentially enhance the model's performance.

Edge Features

The edge features used in this project were the cosine similarity between the entity embedding. However, other edge features, such as the Jaccard similarity [2], could potentially enhance the model's performance.

Hard Negative sampling

Schroff et al. [46] demonstrated that hard negative sampling could enhance the model's performance. This technique could be used to improve the model's performance. This project used a random sampling technique to retrieve a mention that was a negative sample.

Ontology Reasoning

Ontology reasoning could be used to enhance the model's performance. For example, the model could be trained to understand that a person is a type of entity and that a person is a type of human. Consequently, when the model is trained, it will understand that a person is a type of human. Finding negative samples that are semantically similar to the mention could enhance the model's performance.

Pre-training

The entity encoder could be pre-trained on the embedding task based on Knowledge graph embedding benchmark datasets so that the GNN learns structural information based on relationship types. Expanding the datasets to all entity linking dataset both text based and tabular based as a Pre-training task and then using a domain specific dataset to fine tune the model.

7 | Conclusion

The training phase of the model involves interaction solely with entities represented in the mention dataset. As a result, the final positioning of entities not included in the mention dataset remains untrained.

In the process of training dense retrieval models, an assumption is made, as discussed in Section 1.3, that the model's functionality is based on moving mention and entity embeddings that are related closer together, while distancing unrelated ones.

Upon comparing the results in Chapter 5, it is evident that the integration of the Graph Neural Network (GNN) into the entity encoder, based on the model architecture proposed by Wu et al. [2], does not significantly improve the recall 10 validation metric. However, an improvement of 5-10% is observed in recall 50 and recall 100 for each task.

The choice of loss function for training the model also impacts its performance. The model trained with the cross-entropy cosine similarity loss function outperforms the one trained with the triplet margin loss function. This finding aligns with the studies of Louis et al. [24], who compared the triplet margin loss with cross-entropy loss, and Wu et al. [2], who used the dot similarity measurement. However, neither study employed negative sampling during model training.

Chapter 5 reveals that models using bert-base-uncased performed significantly worse than other models. A closer look at the parameters reveals that the bert-tiny model contained approximately 5 million parameters for each encoder, while bert-base-uncased contained about 110 million parameters. Given the size of the datasets and the number of parameters, it is plausible that the model overfitted when using such a large BERT model. This hypothesis is further supported by the loss functions figures in the appendix for large bert models and Table 5.1, where the loss score for bert-base-uncased models was close to zero compared to other models.

To evaluate the performance of the model, we selected four random mentions from the WDC validation dataset and a model with bert-tiny, 5 GNN layers, and trained with cross-entropy cosine loss to examine the 5 entities that were retrieved. Table 7.1 displays the target entity at the top and the retrieved entities in the subsequent five rows. Despite numerous duplicate entities retrieved across the 4 mentions, the MusicAlbum and URL retrieved some entities that are somewhat close.

	Event	**Name**	**MusicAlbum**	**URL**
1	ContentUrl	Genetic	Distance	Genetic
2	dct:identifier	ContentUrl	Eye	ContentUrl
3	Genetic	dct:identifier	dcat:Catalog	dct:identifier
4	entertainmentBusiness	entertainmentBusiness	AlbumRelease	entertainmentBusiness
5	TouristInfomationCenter	chemicalRole	Residence	vehicleSpecialUsage

Table 7.1: *Top 5 entities for each mention based on dense retrieval in the WDC training dataset.*

Table 7.1 indicates that while the model is unable to retrieve the correct entities, the suggested entities are more semantically in nature as opposed to lexicographic similarity. The model is capable of making suggestions based on semantic similarities, demonstrating the feasibility of entity suggestion based on dense retrieval. However, the model requires

further fine-tuning to achieve state-of-the-art performance.

Bibliography

[1] Bossy, Robert et al. 'Bacteria Biotope at BioNLP Open Shared Tasks 2019'. In: *Proceedings of the 5th Workshop on BioNLP Open Shared Tasks*. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 121–131. DOI: 10.18653/v1/D19-5719. URL: https://aclanthology.org/D19-5719 (visited on 09/10/2023).

[2] Wu, Ledell et al. *Scalable Zero-shot Entity Linking with Dense Entity Retrieval*. arXiv:1911.03814 [cs]. Sept. 2020. DOI: 10.48550/arXiv.1911.03814. URL: http://arxiv.org/abs/1911.03814 (visited on 21/08/2023).

[3] Trabelsi, Mohamed et al. 'StruBERT: Structure-aware BERT for Table Search and Matching'. en. In: *Proceedings of the ACM Web Conference 2022*. arXiv:2203.14278 [cs]. Apr. 2022, pp. 442–451. DOI: 10.1145/3485447.3511972. URL: http://arxiv.org/abs/2203.14278 (visited on 17/08/2023).

[4] Devlin, Jacob et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv:1810.04805 [cs]. May 2019. DOI: 10.48550/arXiv.1810.04805. URL: http://arxiv.org/abs/1810.04805 (visited on 01/09/2023).

[5] Wagner, B. and Garcez, A. S. d'Avila. 'Neural-Symbolic Reasoning Under Open-World and Closed-World Assumptions'. en. In: *CEUR Workshop Proceedings*. Vol. 3121. ISSN: 1613-0073. California, USA: CEUR, Mar. 2022. URL: http://ceur-ws.org/Vol-3121/paper3.pdf (visited on 10/10/2023).

[6] Hamilton, William L., Ying, Rex and Leskovec, Jure. *Inductive Representation Learning on Large Graphs*. en. arXiv:1706.02216 [cs, stat]. Sept. 2018. URL: http://arxiv.org/abs/1706.02216 (visited on 09/10/2023).

[7] Goodfellow, Ian, Bengio, Yoshua and Courville, Aaron. *Deep learning*. eng. Adaptive computation and machine learning. Cambridge, Massachusetts London, England: The MIT Press, 2016. ISBN: 978-0-262-03561-3.

[8] Manning, Christopher D., Raghavan, Prabhakar and Schütze, Hinrich. *Introduction to information retrieval*. OCLC: ocn190786122. New York: Cambridge University Press, 2008. ISBN: 978-0-521-86571-5.

[9] Calvanese, Diego et al. 'Ontology-Based Data Access and Integration'. en. In: *Encyclopedia of Database Systems*. Ed. by Liu, Ling and Özsu, M. Tamer. New York, NY: Springer, 2018, pp. 2590–2596. ISBN: 978-1-4614-8265-9. DOI: 10.1007/978-1-4614-8265-9_80667. URL: https://doi.org/10.1007/978-1-4614-8265-9_80667 (visited on 18/10/2023).

[10] *Ontology Matching | SpringerLink*. URL: https://link.springer.com/book/10.1007/978-3-642-38721-0 (visited on 18/10/2023).

[11] Zhou, Jie et al. 'Graph neural networks: A review of methods and applications'. In: *AI Open* 1 (Jan. 2020), pp. 57–81. ISSN: 2666-6510. DOI: 10.1016/j.aiopen.2021.01.001. URL: https://www.sciencedirect.com/science/article/pii/S2666651021000012 (visited on 18/10/2023).

[12] Kipf, Thomas N. and Welling, Max. *Semi-Supervised Classification with Graph Convolutional Networks*. en. arXiv:1609.02907 [cs, stat]. Feb. 2017. URL: http://arxiv.org/abs/1609.02907 (visited on 09/10/2023).

[13] Goyal, Palash and Ferrara, Emilio. 'Graph Embedding Techniques, Applications, and Performance: A Survey'. In: *Knowledge-Based Systems* 151 (July 2018). arXiv:1705.02801 [physics], pp. 78–94. ISSN: 09507051. DOI: 10.1016/j.knosys.2018.03.022. URL: http://arxiv.org/abs/1705.02801 (visited on 18/10/2023).

[14] Gilmer, Justin et al. *Neural Message Passing for Quantum Chemistry*. arXiv:1704.01212 [cs]. June 2017. DOI: 10.48550/arXiv.1704.01212. URL: http://arxiv.org/abs/1704.01212 (visited on 18/10/2023).

[15] Hadsell, R., Chopra, S. and LeCun, Y. 'Dimensionality Reduction by Learning an Invariant Mapping'. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. ISSN: 1063-6919. June 2006, pp. 1735–1742. DOI: 10.1109/CVPR.2006.100.

[16] Karpukhin, Vladimir et al. 'Dense Passage Retrieval for Open-Domain Question Answering'. In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Online: Association for Computational Linguistics, Nov. 2020, pp. 6769–6781. DOI: 10.18653/v1/2020.emnlp-main.550. URL: https://aclanthology.org/2020.emnlp-main.550 (visited on 18/10/2023).

[17] Rekabsaz, Navid et al. 'Volatility Prediction using Financial Disclosures Sentiments with Word Embedding-based IR Models'. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, July 2017, pp. 1712–1721. DOI: 10.18653/v1/P17-1157. URL: https://aclanthology.org/P17-1157 (visited on 18/10/2023).

[18] Shen, Wei, Wang, Jianyong and Han, Jiawei. 'Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions'. In: *IEEE Transactions on Knowledge and Data Engineering* 27.2 (Feb. 2015). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 443–460. ISSN: 1558-2191. DOI: 10.1109/TKDE.2014.2327028. URL: https://ieeexplore.ieee.org/document/6823700 (visited on 18/10/2023).

[19] Mendes, Pablo et al. 'DBpedia Spotlight: Shedding Light on the Web of Documents'. In: *Proceedings of the 7th International Conference on Semantic Systems* (Sept. 2011), pp. 1–8. DOI: 10.1145/2063518.2063519. URL: https://corescholar.libraries.wright.edu/knoesis/1034.

[20] Rao, Delip, McNamee, Paul and Dredze, Mark. 'Entity Linking: Finding Extracted Entities in a Knowledge Base'. en. In: *Multi-source, Multilingual Information Extraction and Summarization*. Ed. by Poibeau, Thierry et al. Theory and Applications of Natural Language Processing. Berlin, Heidelberg: Springer, 2013, pp. 93–115. ISBN: 978-3-642-28569-1. DOI: 10.1007/978-3-642-28569-1_5. URL: https://doi.org/10.1007/978-3-642-28569-1_5 (visited on 18/10/2023).

[21] Dredze, Mark et al. 'Entity Disambiguation for Knowledge Base Population'. In: *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*. Beijing, China: Coling 2010 Organizing Committee, Aug. 2010, pp. 277–285. URL: https://aclanthology.org/C10-1032 (visited on 21/08/2023).

[22] He, Zhengyan et al. 'Learning Entity Representation for Entity Disambiguation'. In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 30–34. URL: https://aclanthology.org/P13-2006 (visited on 21/08/2023).

[23] Wang, Zhen et al. 'Knowledge Graph and Text Jointly Embedding'. en. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014, pp. 1591–1601. DOI: 10.3115/v1/D14-1167. URL: http://aclweb.org/anthology/D14-1167 (visited on 17/08/2023).

[24] Louis, Antoine, Dijck, Gijs van and Spanakis, Gerasimos. 'Finding the Law: Enhancing Statutory Article Retrieval via Graph Neural Networks'. In: *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*. Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 2761–2776. DOI: 10.18653/v1/2023.eacl-main.203. URL: https://aclanthology.org/2023.eacl-main.203 (visited on 09/10/2023).

[25] Nédellec, Claire. *OntoBiotope-BioNLP-OST*. en. Oct. 2023. DOI: 10.15454/VSQKHB. URL: https://entrepot.recherche.data.gouv.fr/dataset.xhtml?persistentId=doi:10.15454/VSQKHB (visited on 09/10/2023).

[26] Jackson, Rebecca et al. 'OBO Foundry in 2021: operationalizing open data principles to evaluate ontologies'. In: *Database* 2021 (Oct. 2021). _eprint: https://academic.oup.com/database/article-pdf/doi/10.1093/database/baab069/40854912/baab069.pdf, baab069. ISSN: 1758-0463. DOI: 10.1093/database/baab069. URL: https://doi.org/10.1093/database/baab069.

[27] Ritze, Dominique and Bizer, Christian. *Matching Web Tables To DBpedia - A Feature Utility Study*. en. 2017. DOI: 10.5441/002/EDBT.2017.20. URL: https://openproceedings.org/2017/conf/edbt/paper-148.pdf (visited on 09/10/2023).

[28] Oliveira, Daniela and Pesquita, Catia. *SemTab 2021 BioTable Dataset*. Oct. 2021. DOI: 10.5281/zenodo.5606585. URL: https://zenodo.org/record/5606585 (visited on 09/10/2023).

[29] Cutrona, Vincenzo et al. 'Tough Tables: Carefully Evaluating Entity Linking for Tabular Data'. en. In: *The Semantic Web – ISWC 2020*. Ed. by Pan, Jeff Z. et al. Vol. 12507. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 328–343. ISBN: 978-3-030-62465-1. DOI: 10.1007/978-3-030-62466-8_21. URL: https://link.springer.com/10.1007/978-3-030-62466-8_21 (visited on 09/10/2023).

[30] Hulsebos, Madelon, Demiralp, Çağatay and Groth, Paul. 'GitTables: A Large-Scale Corpus of Relational Tables'. en. In: *Proceedings of the ACM on Management of Data* 1.1 (May 2023). arXiv:2106.07258 [cs], pp. 1–17. ISSN: 2836-6573. DOI: 10.1145/3588710. URL: http://arxiv.org/abs/2106.07258 (visited on 09/10/2023).

[31] Efthymiou, Vasilis et al. 'Matching Web Tables with Knowledge Base Entities: From Entity Lookups to Entity Embeddings'. en. In: *The Semantic Web – ISWC 2017*. Ed. by d'Amato, Claudia et al. Vol. 10587. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 260–277. ISBN: 978-3-319-68287-7. DOI: 10.1007/978-3-319-68288-4_16. URL: https://link.springer.com/10.1007/978-3-319-68288-4_16 (visited on 09/10/2023).

[32] Zhang, Dan et al. *Sato: Contextual Semantic Type Detection in Tables*. en. arXiv:1911.06311 [cs]. June 2020. URL: http://arxiv.org/abs/1911.06311 (visited on 09/10/2023).

[33] Abdallah, Abdelrahman et al. 'TNCR: Table Net Detection and Classification Dataset'. In: *Neurocomputing* 473 (Feb. 2022). arXiv:2106.15322 [cs], pp. 79–97. ISSN: 09252312. DOI: 10.1016/j.neucom.2021.11.101. URL: http://arxiv.org/abs/2106.15322 (visited on 09/10/2023).

[34] Deng, Xiang et al. *TURL: Table Understanding through Representation Learning*. en. arXiv:2006.14806 [cs]. Dec. 2020. URL: http://arxiv.org/abs/2006.14806 (visited on 09/10/2023).

[35] Korini, Keti. 'SOTAB: The WDC Schema.org Table Annotation Benchmark'. en. In: ().

[36] Veyhe, Bartal Eyðfinsson, Sagi, Tomer and Hose, Katja. 'Scientific Data Extraction from Oceanographic Papers'. In: *ACM Web Conference 2023 - Companion of the World Wide Web Conference, WWW 2023* (Apr. 2023). Publisher: Association for Computing Machinery, pp. 800–804. DOI: 10.1145/3543873.3587595. URL: http://www.scopus.com/inward/record.url?scp=85159581806&partnerID=8YFLogxK (visited on 13/10/2023).

[37] *Wikidata:Database reports/List of properties/all - Wikidata*. URL: https://www.wikidata.org/wiki/Wikidata:Database_reports/List_of_properties/all (visited on 09/10/2023).

[38] Dong, Zhe et al. *Exploring Dual Encoder Architectures for Question Answering*. en. Apr. 2022. URL: https://arxiv.org/abs/2204.07120v2 (visited on 09/10/2023).

[39] Team, Keras. *Keras documentation: Natural language image search with a Dual Encoder*. en. URL: https://keras.io/examples/vision/nl_image_search/ (visited on 09/10/2023).

[40] Yang, Yulei and Li, Dongsheng. 'NENN: Incorporate Node and Edge Features in Graph Neural Networks'. en. In: *Proceedings of The 12th Asian Conference on Machine Learning*. ISSN: 2640-3498. PMLR, Sept. 2020, pp. 593–608. URL: https://proceedings.mlr.press/v129/yang20a.html (visited on 10/10/2023).

[41] Gong, Liyu and Cheng, Qiang. *Exploiting Edge Features in Graph Neural Networks*. en. arXiv:1809.02709 [cs, stat]. Jan. 2019. URL: http://arxiv.org/abs/1809.02709 (visited on 10/10/2023).

[42] Veličković, Petar et al. *Graph Attention Networks*. en. arXiv:1710.10903 [cs, stat]. Feb. 2018. URL: http://arxiv.org/abs/1710.10903 (visited on 09/10/2023).

[43] Morris, Christopher et al. 'Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks'. en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 4602–4609. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v33i01.33014602. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4384 (visited on 09/10/2023).

[44] Brody, Shaked, Alon, Uri and Yahav, Eran. *How Attentive are Graph Attention Networks?* en. arXiv:2105.14491 [cs]. Jan. 2022. URL: http://arxiv.org/abs/2105.14491 (visited on 09/10/2023).

[45] Johnson, Jeff, Douze, Matthijs and Jégou, Hervé. *Billion-scale similarity search with GPUs*. arXiv:1702.08734 [cs]. Feb. 2017. DOI: 10.48550/arXiv.1702.08734. URL: http://arxiv.org/abs/1702.08734 (visited on 10/10/2023).

[46] Schroff, Florian, Kalenichenko, Dmitry and Philbin, James. *FaceNet: A Unified Embedding for Face Recognition and Clustering*. en. Mar. 2015. DOI: 10.1109/CVPR.2015.7298682. URL: https://arxiv.org/abs/1503.03832v3 (visited on 10/10/2023).

[47] Scarselli, F. et al. 'The Graph Neural Network Model'. In: *IEEE Transactions on Neural Networks* 20.1 (Jan. 2009), pp. 61–80. ISSN: 1045-9227, 1941-0093. DOI: 10.1109/TNN.2008.2005605. URL: http://ieeexplore.ieee.org/document/4700287/ (visited on 01/09/2023).

[48] Liu, Yinhan et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv:1907.11692 [cs]. July 2019. DOI: 10.48550/arXiv.1907.11692. URL: http://arxiv.org/abs/1907.11692 (visited on 18/10/2023).

[49] Abdelmageed, Nora et al. 'Results of SemTab 2022'. In: *Semantic Web Challenge on Tabular Data to Knowledge Graph Matching* 3320 (2022). Publisher: CEUR.

Appendices

A | Model runs charts

bert-large 0 Schema Tripplet margin loss

For the dataset Schema, we used the Bi-directional bert encoder with 0 GNN layers and the bert-large model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

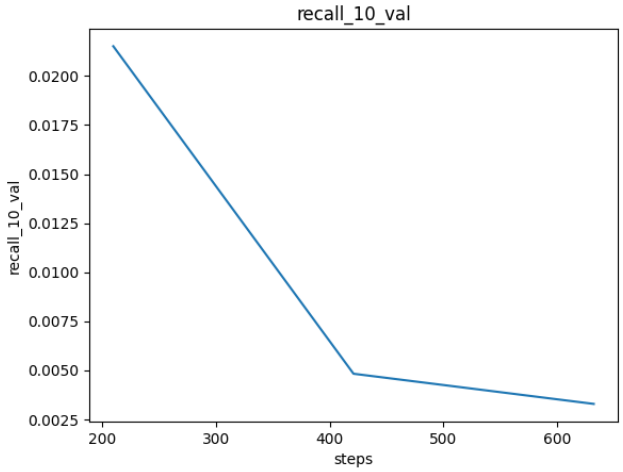


Figure A.1: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

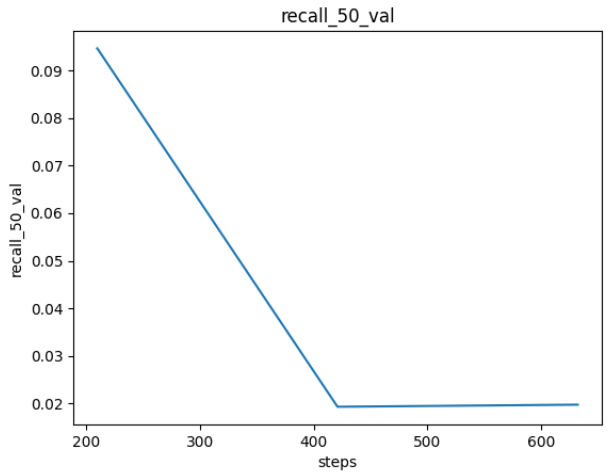


Figure A.2: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

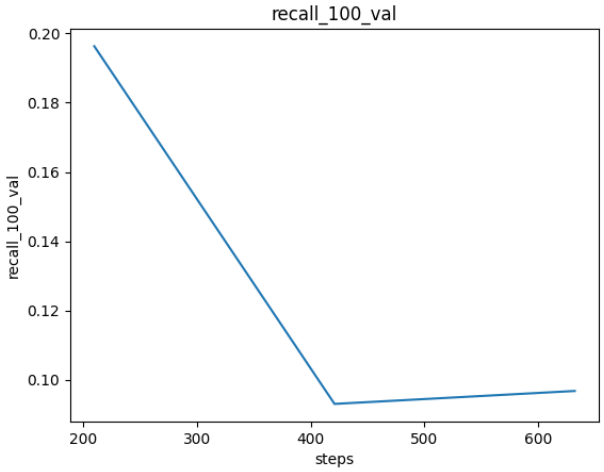


Figure A.3: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

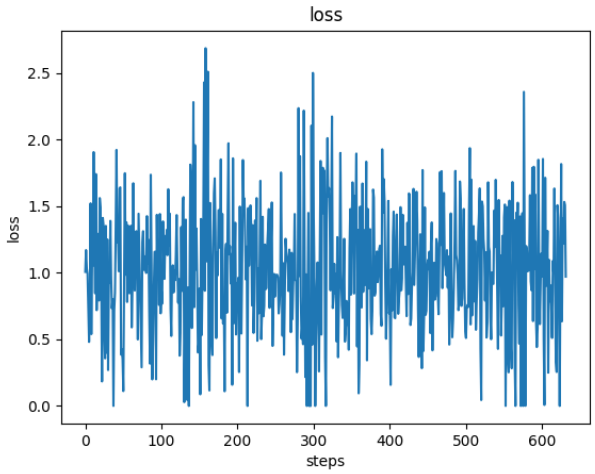


Figure A.4: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

bert-large 5 BB Tripplet margin loss

For the dataset BB, we used the Bi-directional bert encoder with 5 GNN layers and the bert-large model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

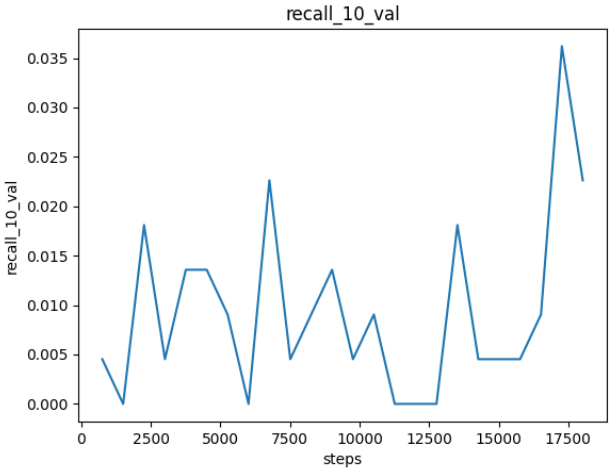


Figure A.5: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

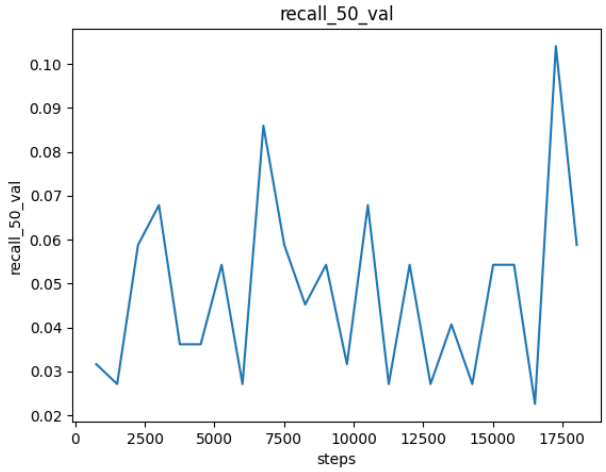


Figure A.6: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

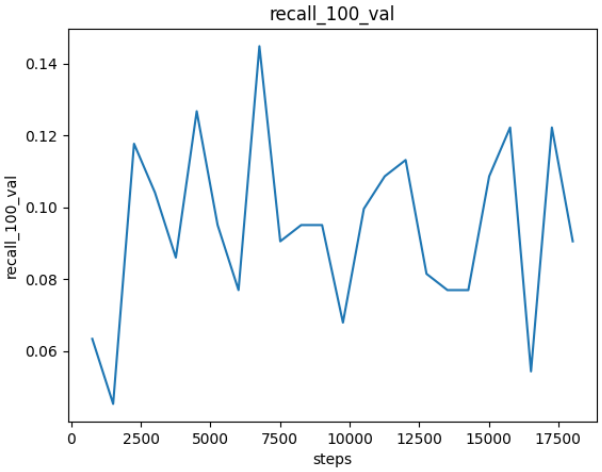


Figure A.7: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

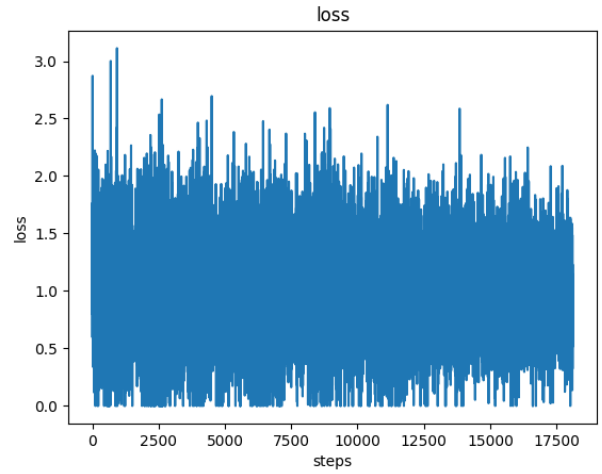


Figure A.8: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

bert-large 3 BB Tripplet margin loss

For the dataset BB, we used the Bi-directional bert encoder with 3 GNN layers and the bert-large model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

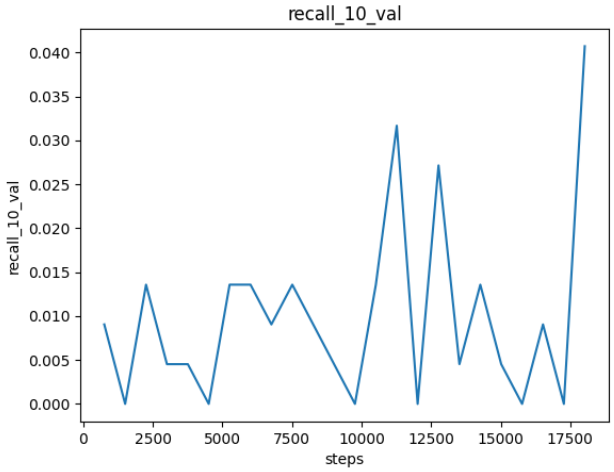


Figure A.9: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

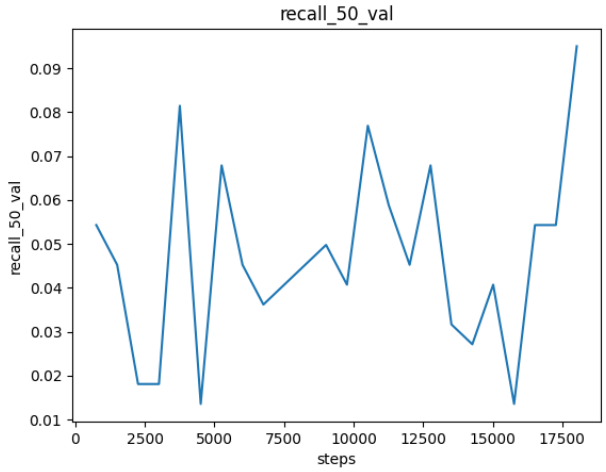


Figure A.10: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

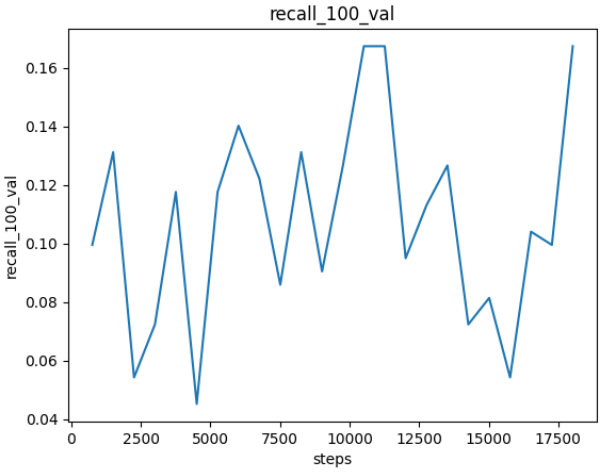


Figure A.11: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

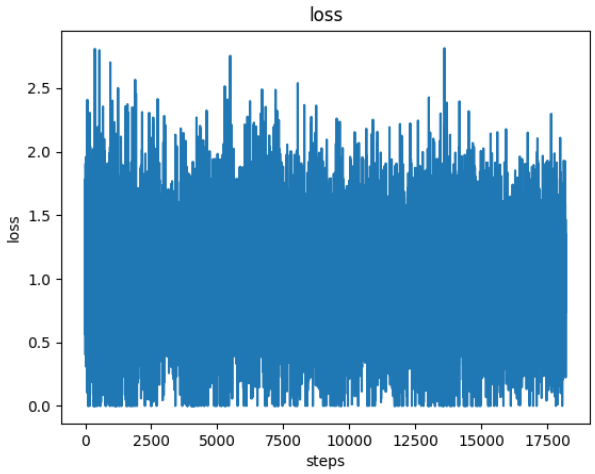


Figure A.12: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

bert-large 0 BB Tripplet margin loss

For the dataset BB, we used the Bi-directional bert encoder with 0 GNN layers and the bert-large model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

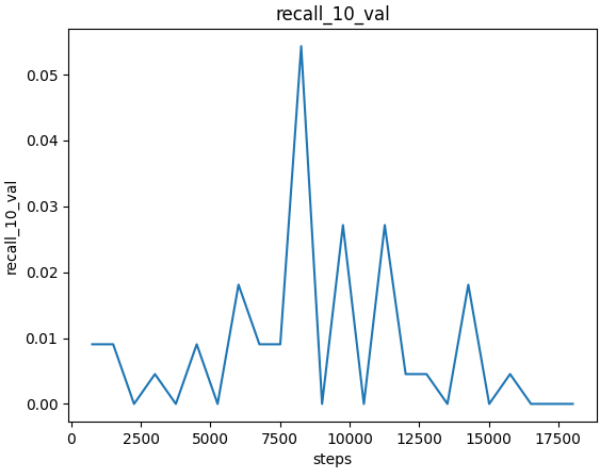


Figure A.13: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

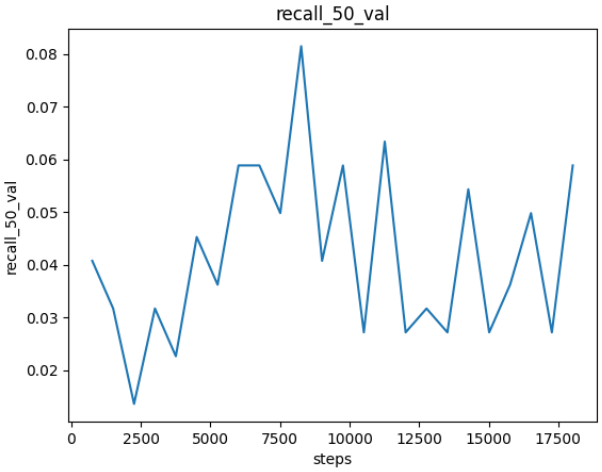


Figure A.14: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

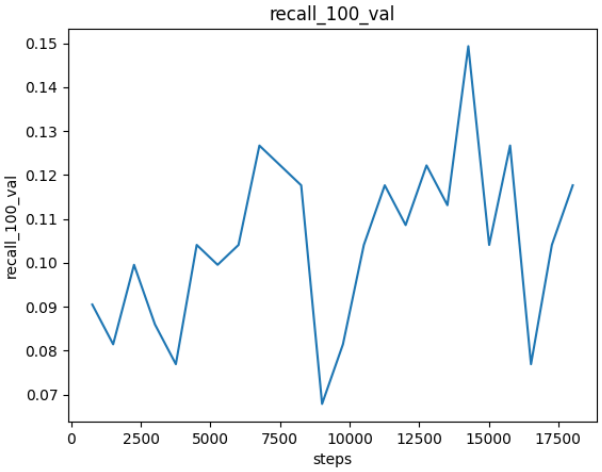


Figure A.15: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

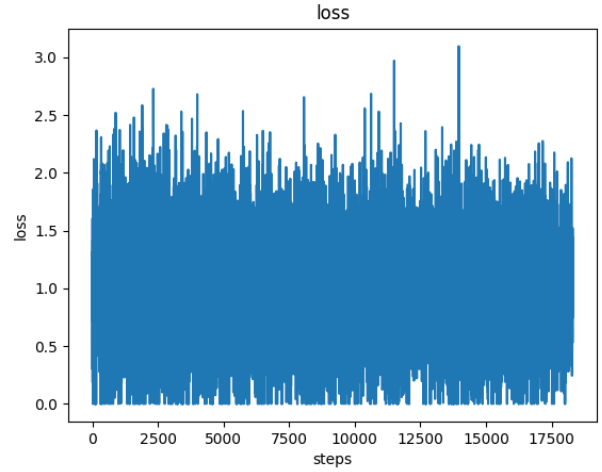


Figure A.16: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

bert-large 5 DBpeida cosine

For the dataset DBpeida, we used the Bi-directional bert encoder with 5 GNN layers and the bert-large model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

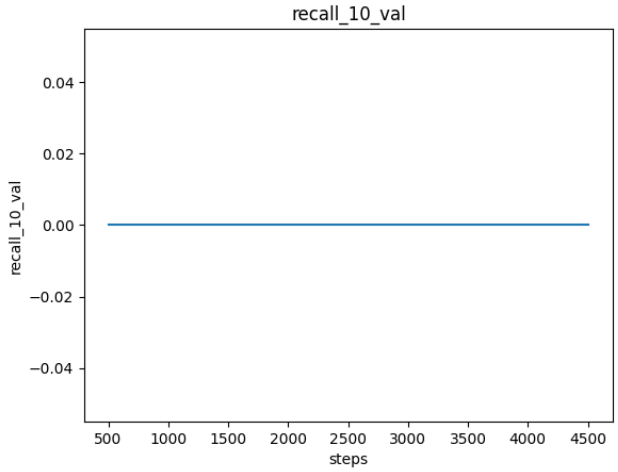


Figure A.17: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

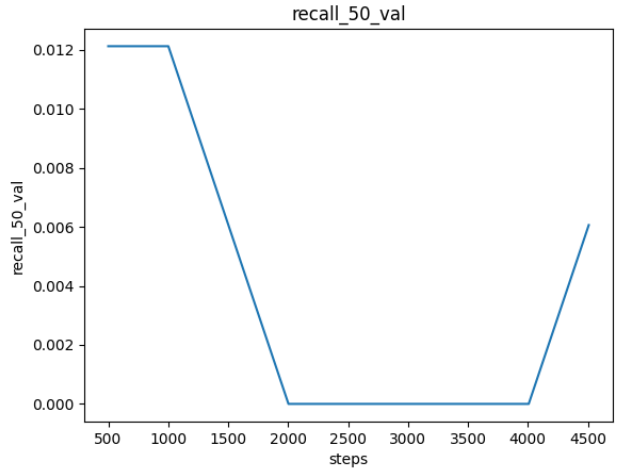


Figure A.18: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

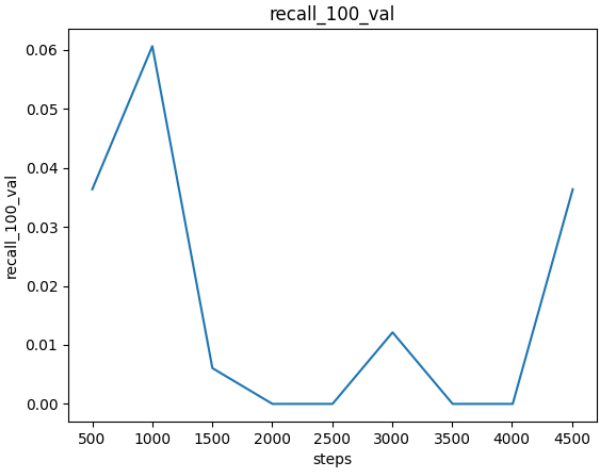


Figure A.19: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

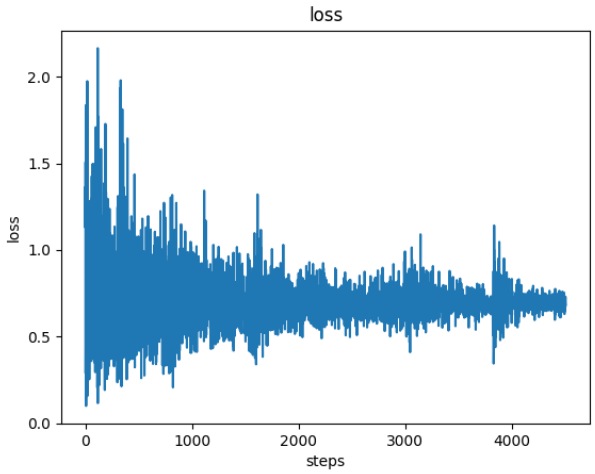


Figure A.20: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

bert-large 3 DBpeida cosine

For the dataset DBpeida, we used the Bi-directional bert encoder with 3 GNN layers and the bert-large model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

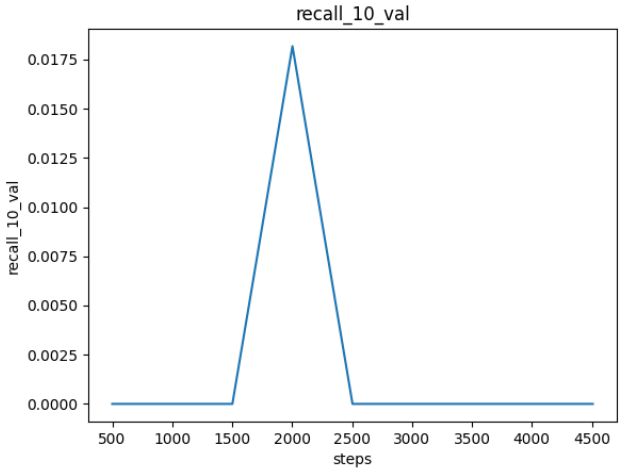


Figure A.21: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

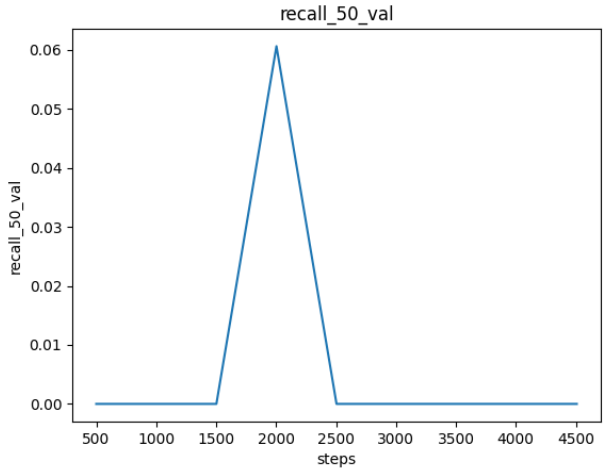


Figure A.22: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

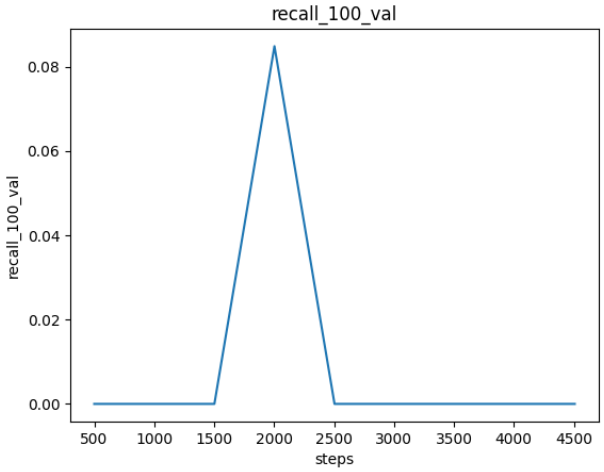


Figure A.23: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

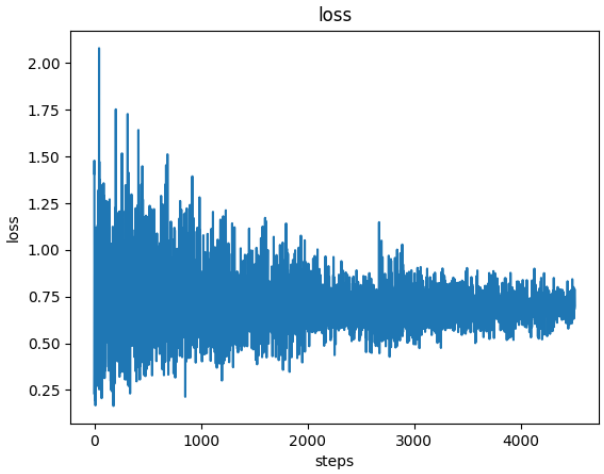


Figure A.24: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

bert-large 0 DBpeida cosine

For the dataset DBpeida, we used the Bi-directional bert encoder with 0 GNN layers and the bert-large model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

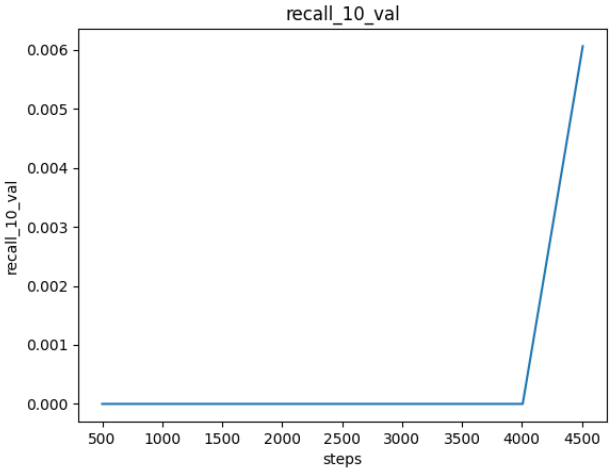


Figure A.25: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

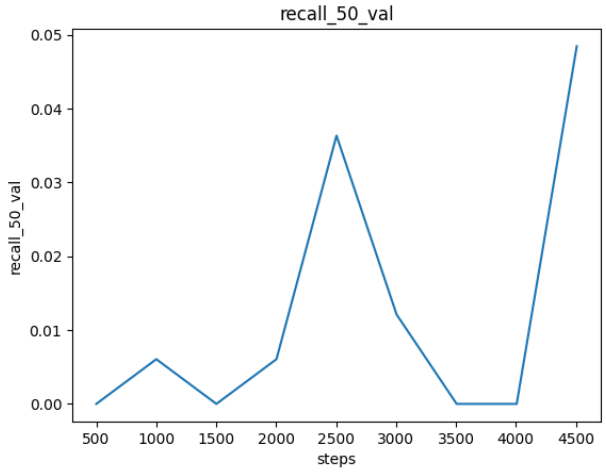


Figure A.26: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

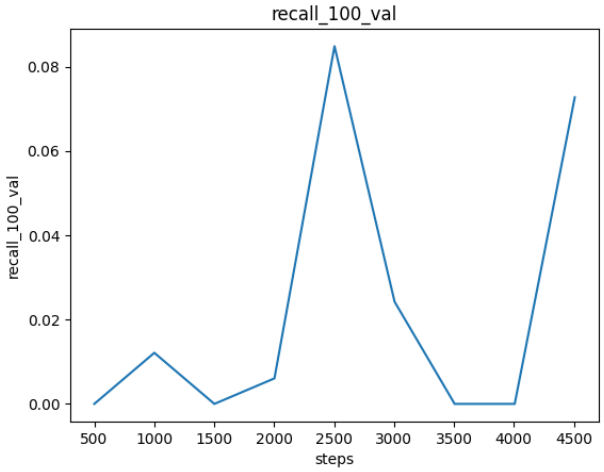


Figure A.27: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

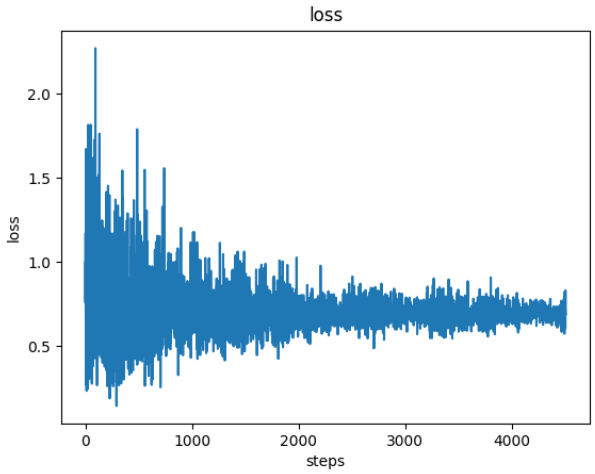


Figure A.28: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

bert 5 DBpeida cosine

For the dataset DBpeida, we used the Bi-directional bert encoder with 5 GNN layers and the bert model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

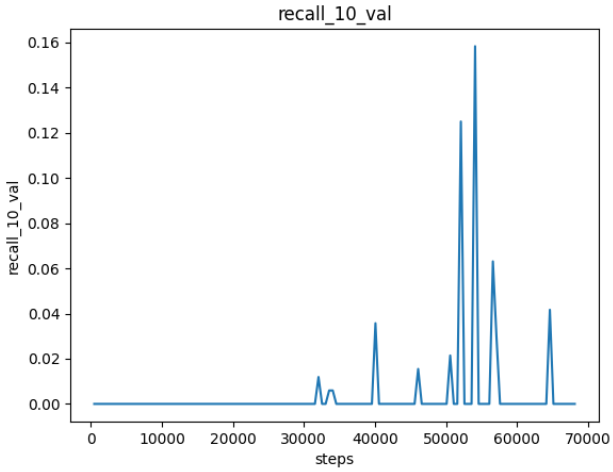


Figure A.29: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

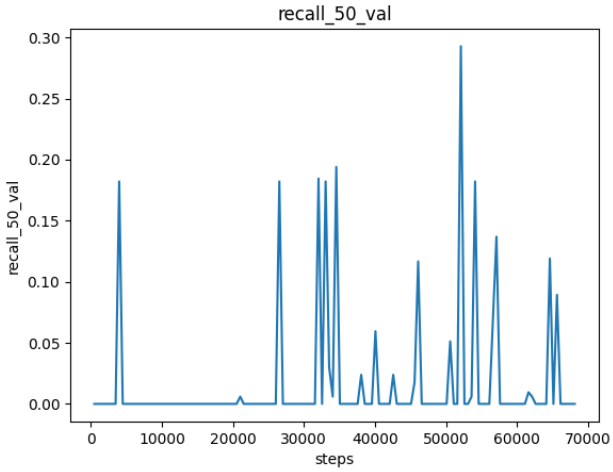


Figure A.30: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

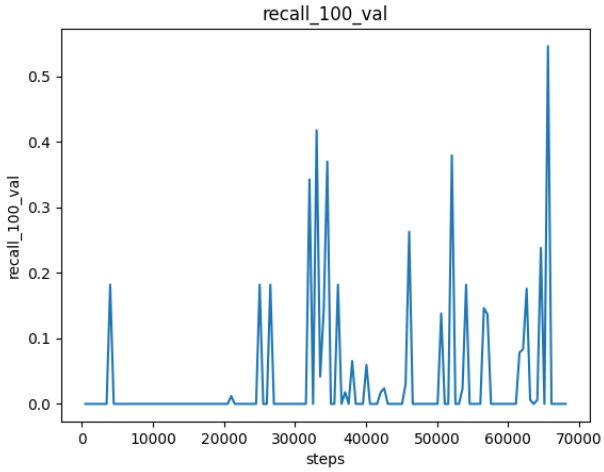


Figure A.31: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

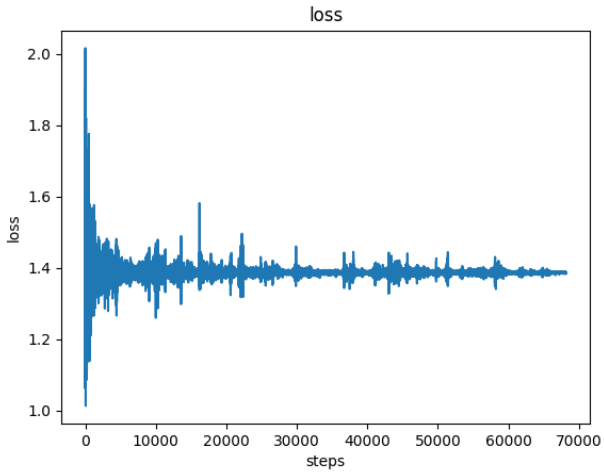


Figure A.32: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

bert 3 DBpeida cosine

For the dataset DBpeida, we used the Bi-directional bert encoder with 3 GNN layers and the bert model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

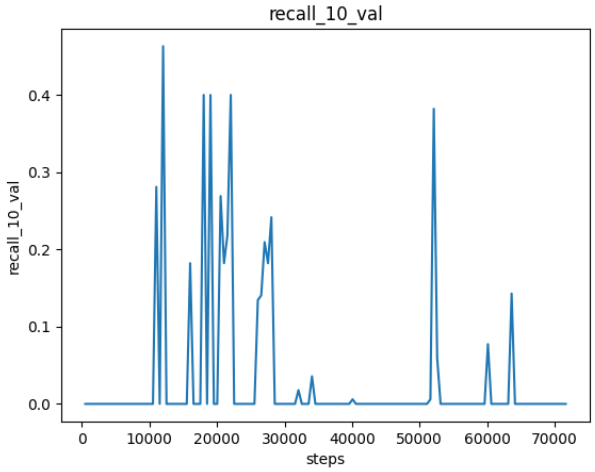


Figure A.33: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

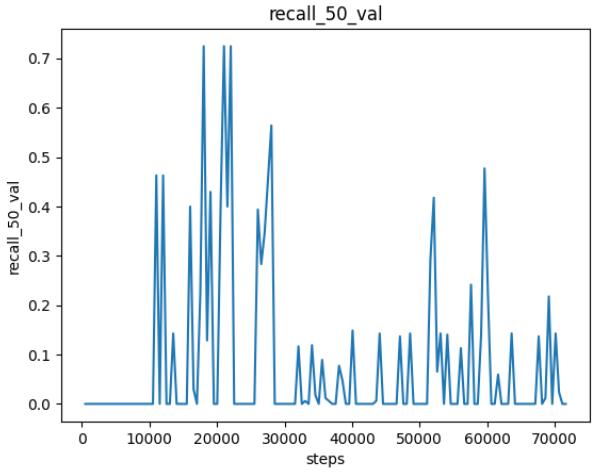


Figure A.34: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

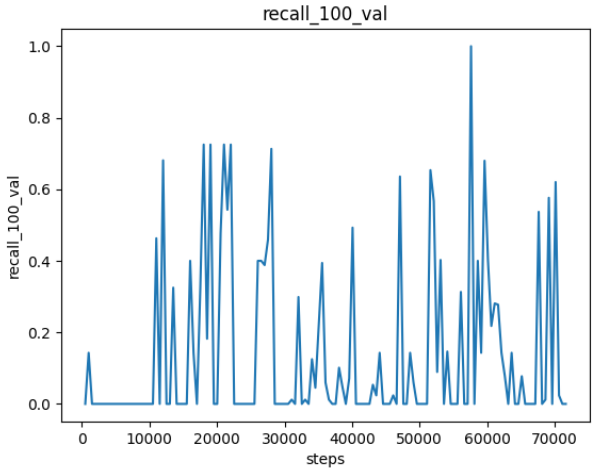


Figure A.35: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

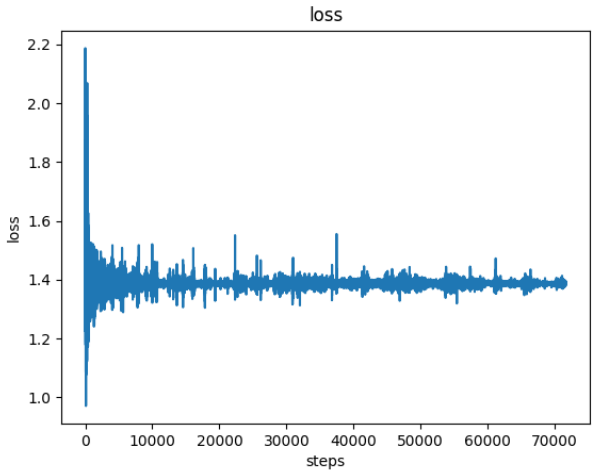


Figure A.36: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

bert-large 3 Schema cosine

For the dataset Schema, we used the Bi-directional bert encoder with 3 GNN layers and the bert-large model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

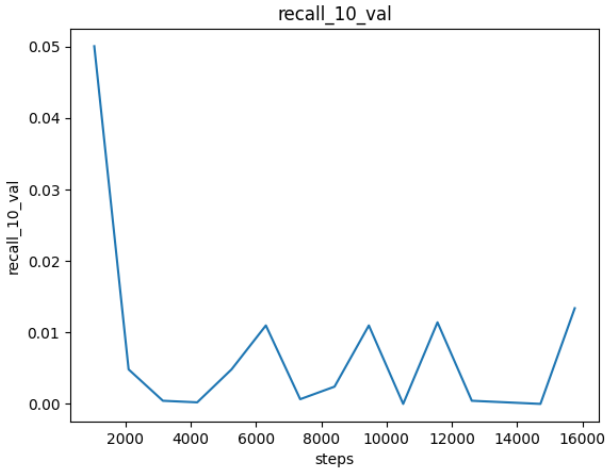


Figure A.37: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

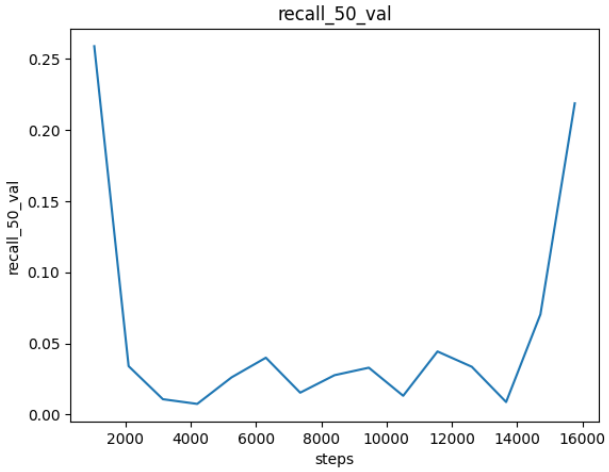


Figure A.38: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

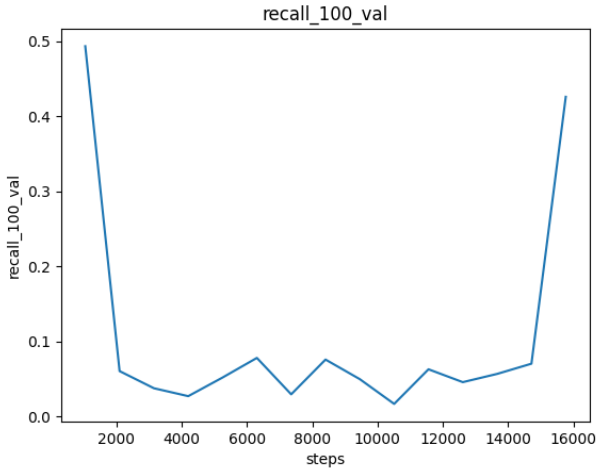


Figure A.39: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

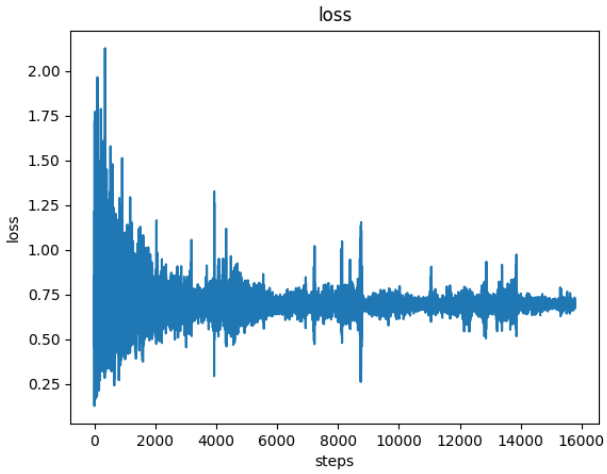


Figure A.40: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

bert-large 5 Schema cosine

For the dataset Schema, we used the Bi-directional bert encoder with 5 GNN layers and the bert-large model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

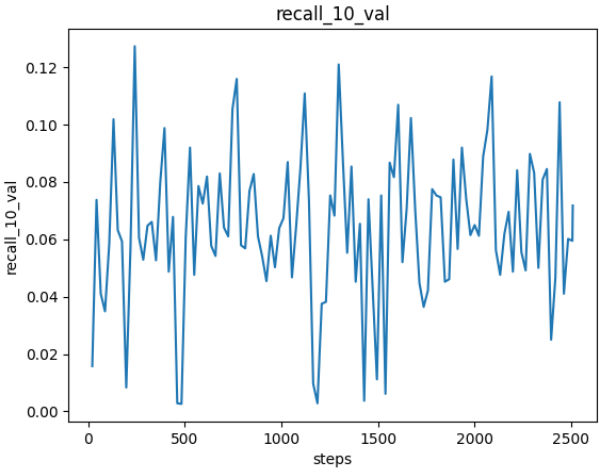


Figure A.41: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

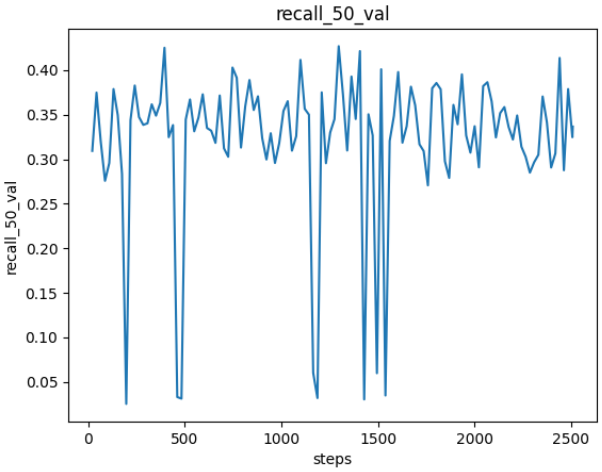


Figure A.42: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

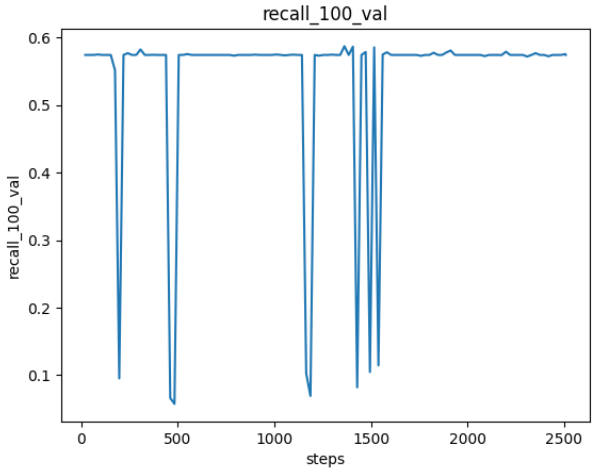


Figure A.43: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

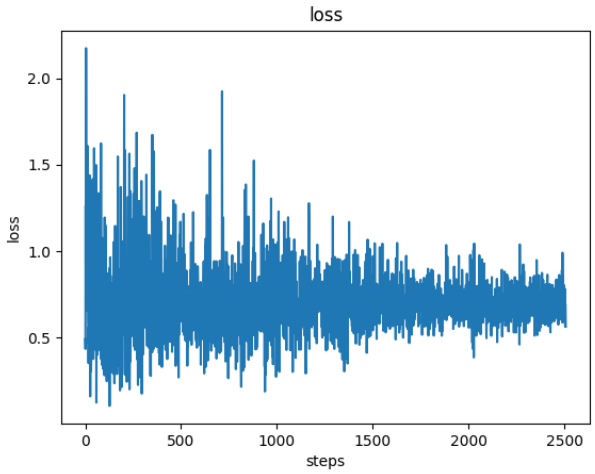


Figure A.44: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

bert-large 5 Schema cosine

For the dataset Schema, we used the Bi-directional bert encoder with 5 GNN layers and the bert-large model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

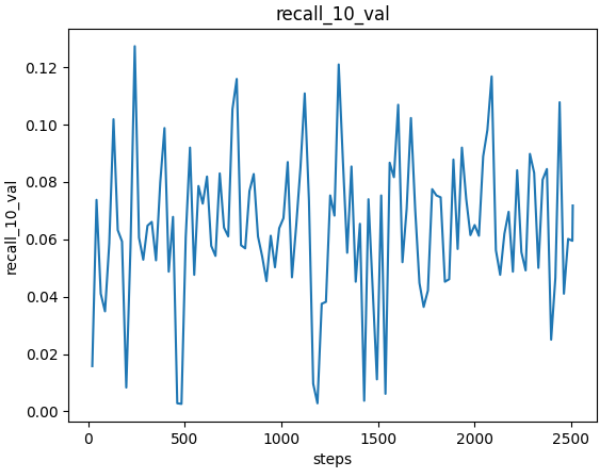


Figure A.45: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

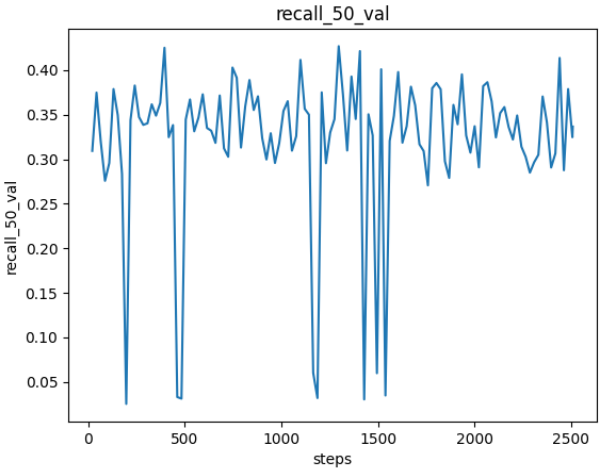


Figure A.46: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

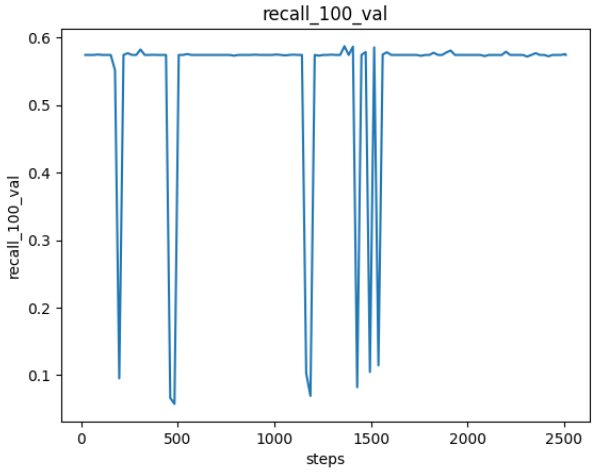


Figure A.47: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

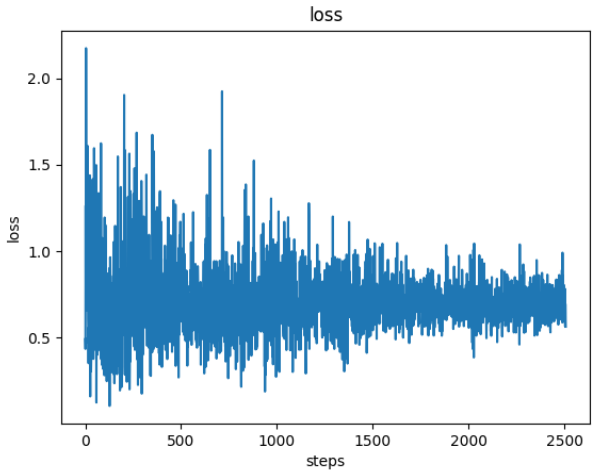


Figure A.48: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

bert-large 0 Schema cosine

For the dataset Schema, we used the Bi-directional bert encoder with 0 GNN layers and the bert-large model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

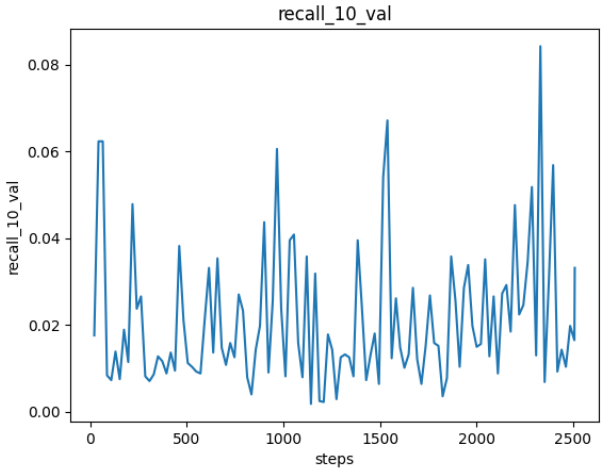


Figure A.49: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

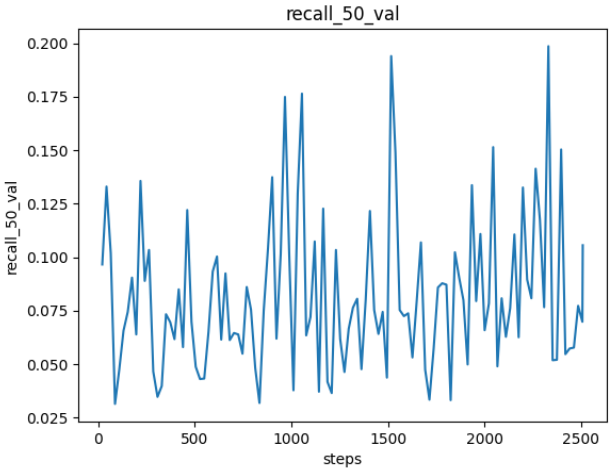


Figure A.50: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

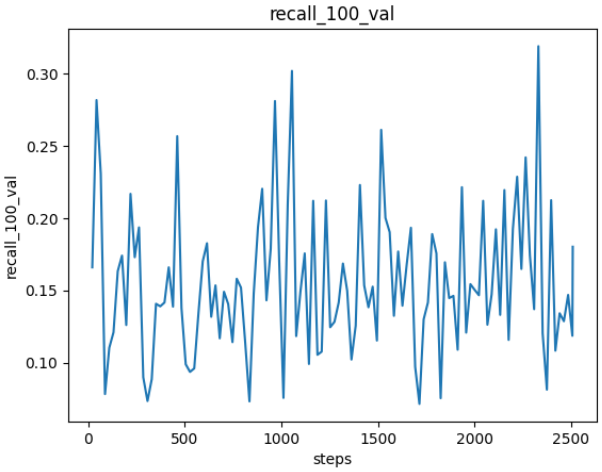


Figure A.51: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

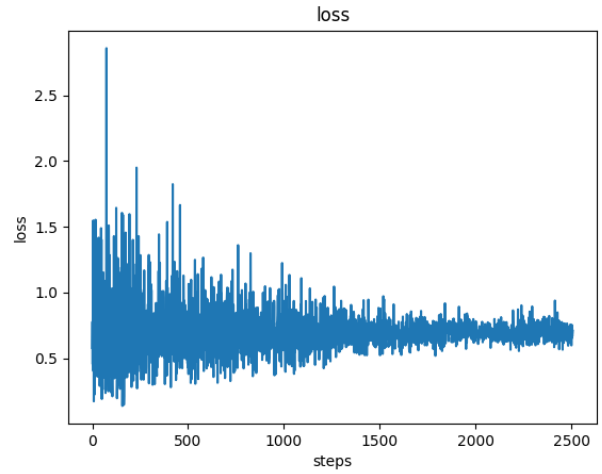


Figure A.52: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

bert-large 3 BB cosine

For the dataset BB, we used the Bi-directional bert encoder with 3 GNN layers and the bert-large model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

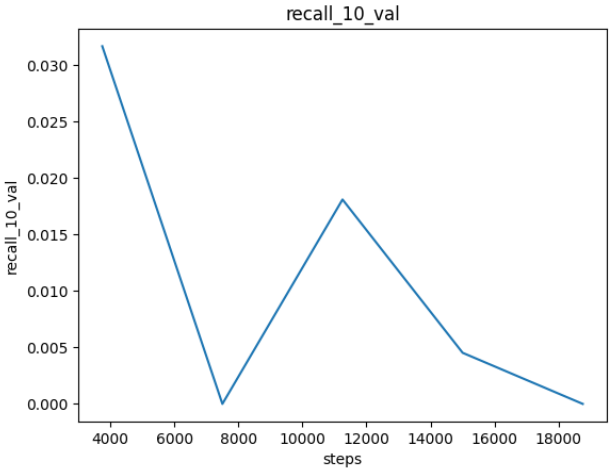


Figure A.53: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

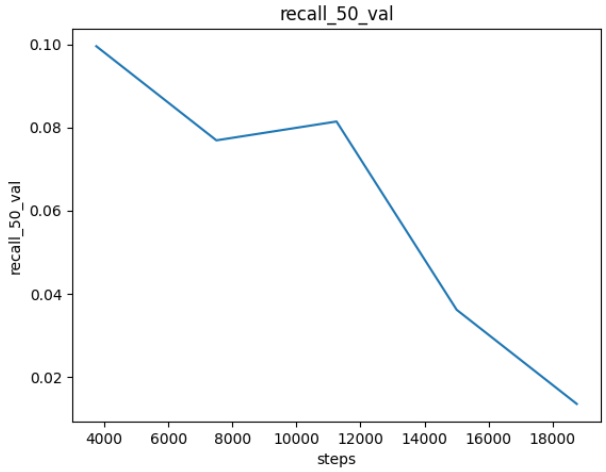


Figure A.54: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

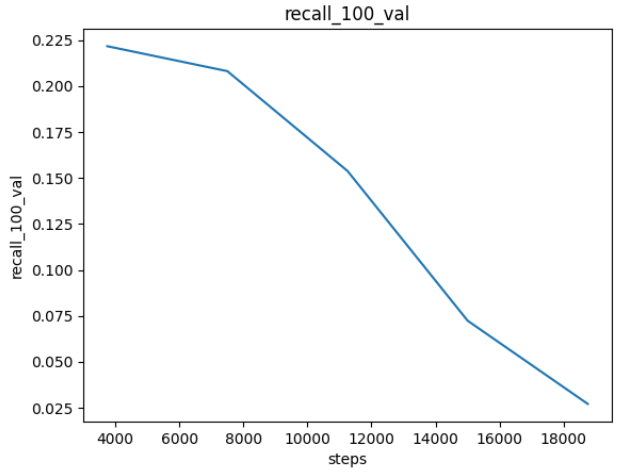


Figure A.55: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

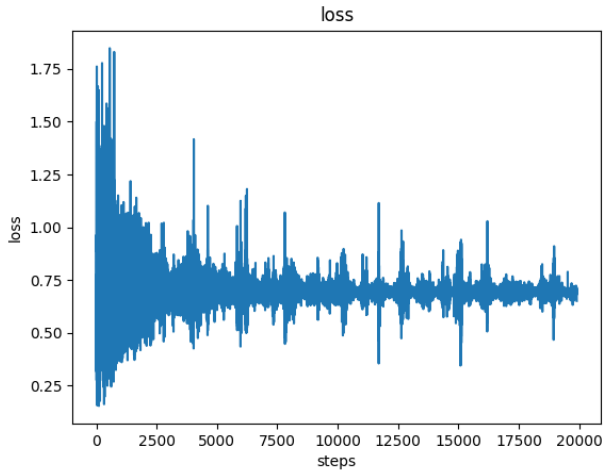


Figure A.56: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

bert-large 5 BB cosine

For the dataset BB, we used the Bi-directional bert encoder with 5 GNN layers and the bert-large model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

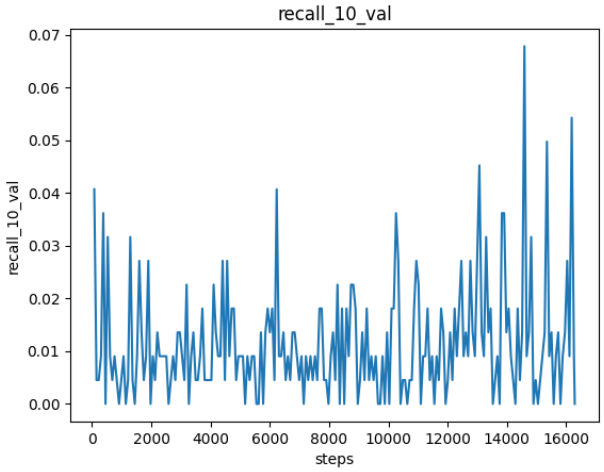


Figure A.57: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

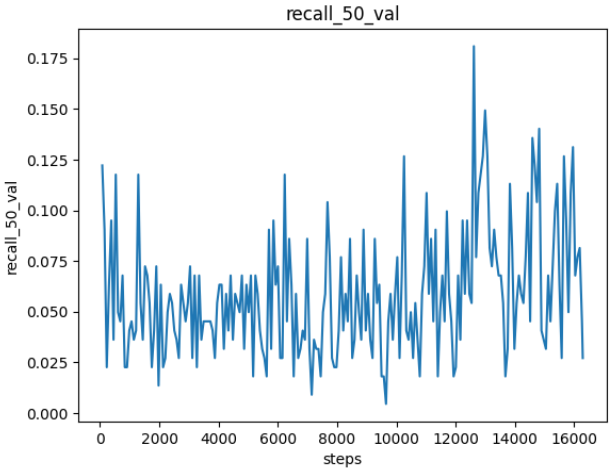


Figure A.58: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

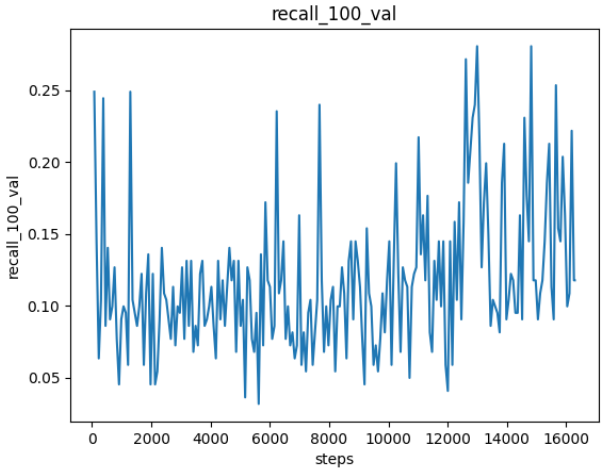


Figure A.59: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

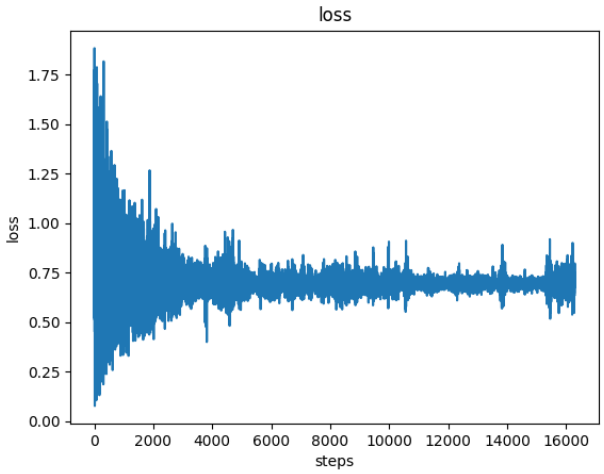


Figure A.60: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

bert-large 0 BB cosine

For the dataset BB, we used the Bi-directional bert encoder with 0 GNN layers and the bert-large model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

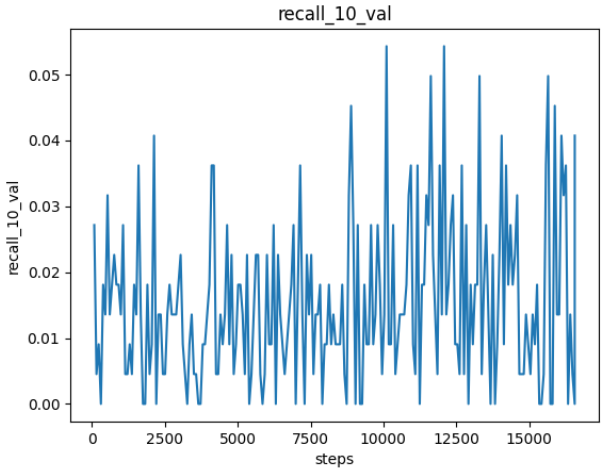


Figure A.61: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

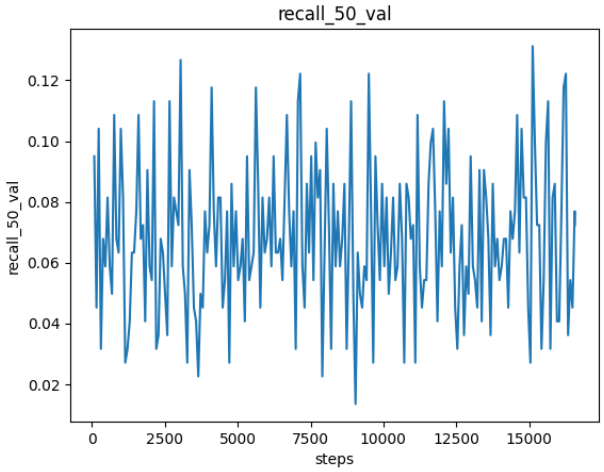


Figure A.62: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

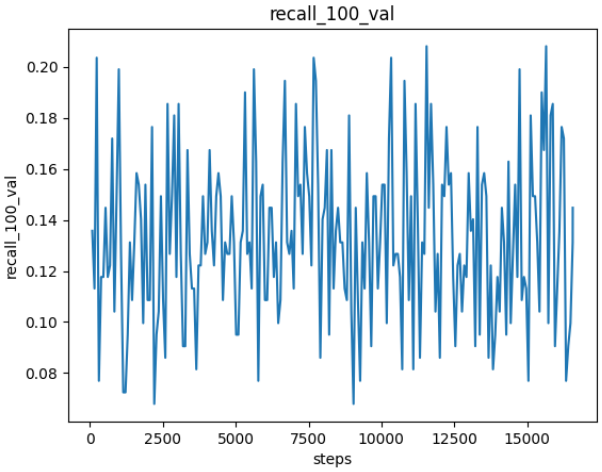


Figure A.63: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

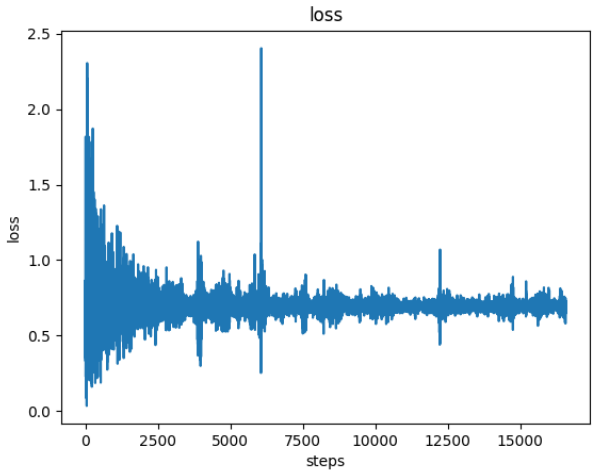


Figure A.64: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert-large, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

bert 0 DBpeida cosine

For the dataset DBpeida, we used the Bi-directional bert encoder with 0 GNN layers and the bert model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

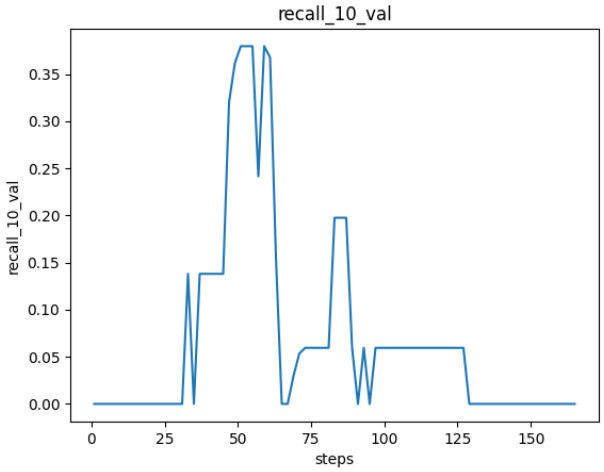


Figure A.65: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

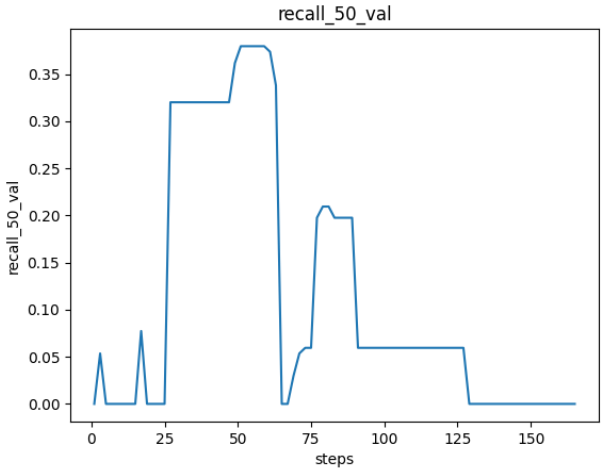


Figure A.66: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

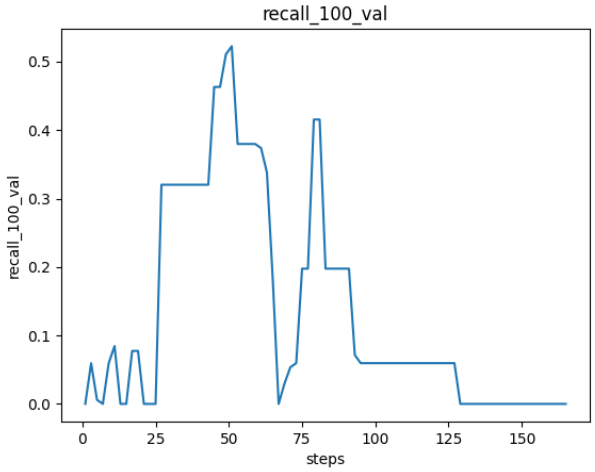


Figure A.67: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

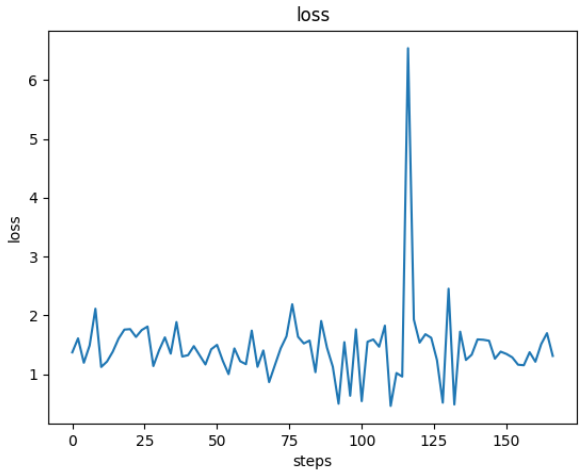


Figure A.68: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

bert 5 Schema cosine

For the dataset Schema, we used the Bi-directional bert encoder with 5 GNN layers and the bert model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

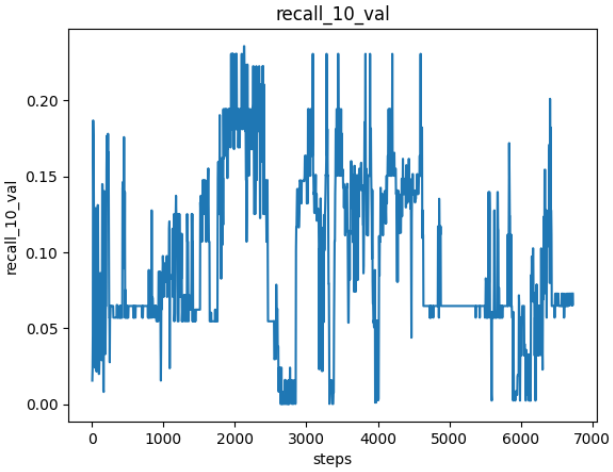


Figure A.69: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

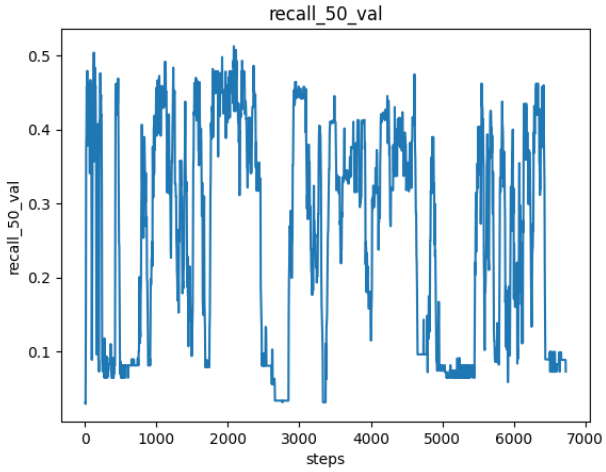


Figure A.70: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

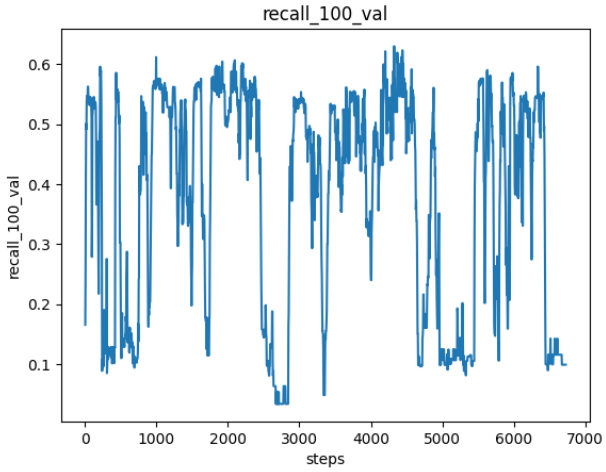


Figure A.71: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

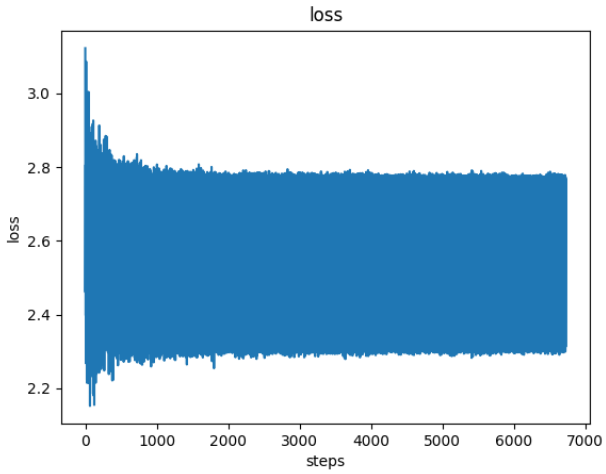


Figure A.72: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

bert 3 Schema cosine

For the dataset Schema, we used the Bi-directional bert encoder with 3 GNN layers and the bert model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

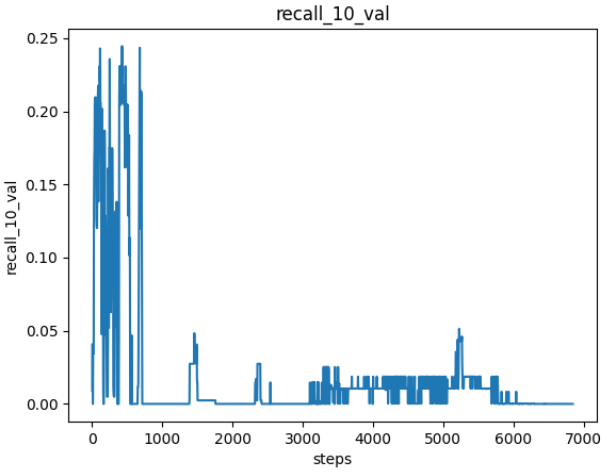


Figure A.73: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

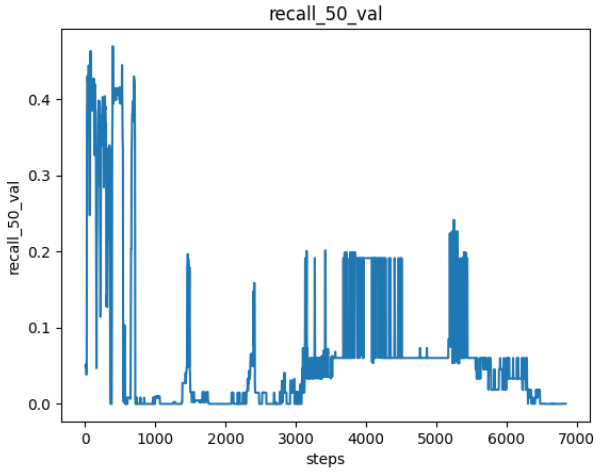


Figure A.74: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

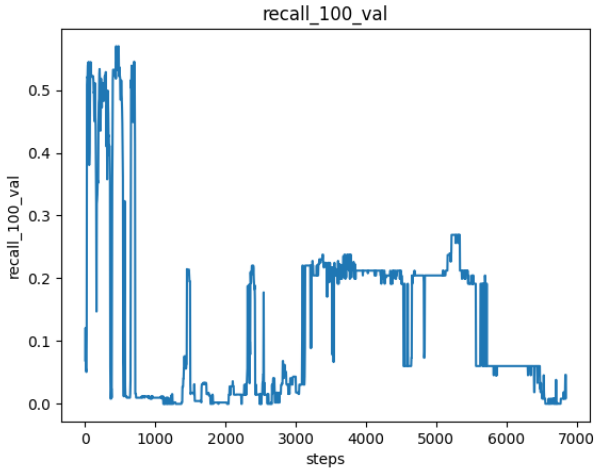


Figure A.75: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

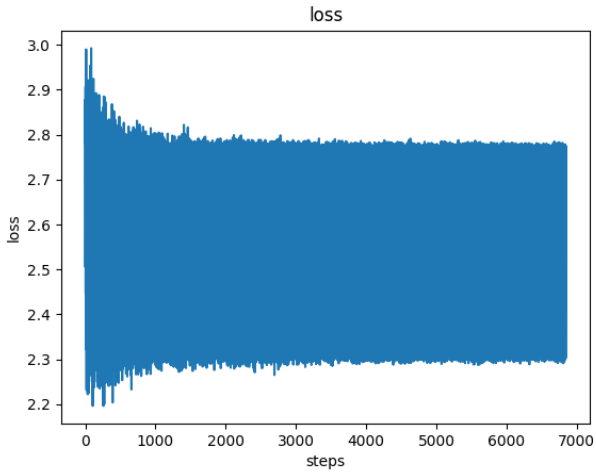


Figure A.76: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

bert 0 Schema cosine

For the dataset Schema, we used the Bi-directional bert encoder with 0 GNN layers and the bert model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

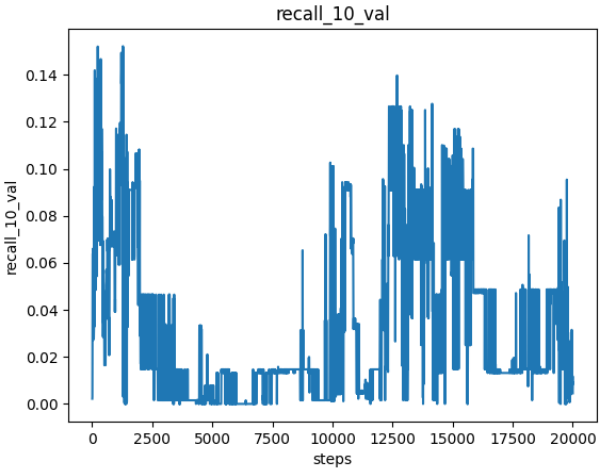


Figure A.77: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

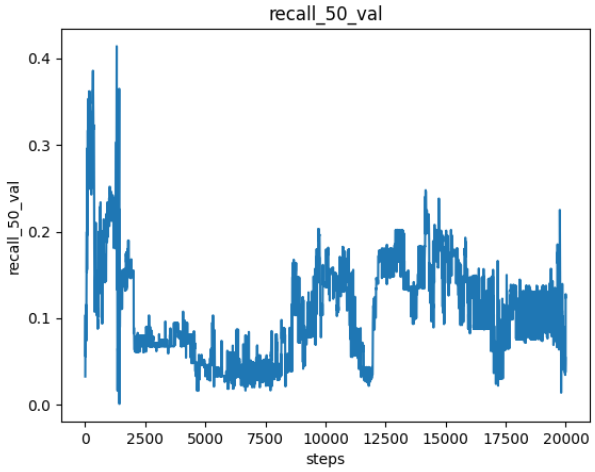


Figure A.78: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

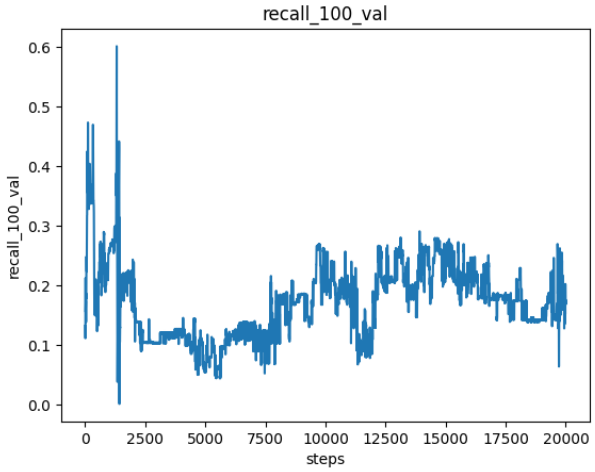


Figure A.79: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

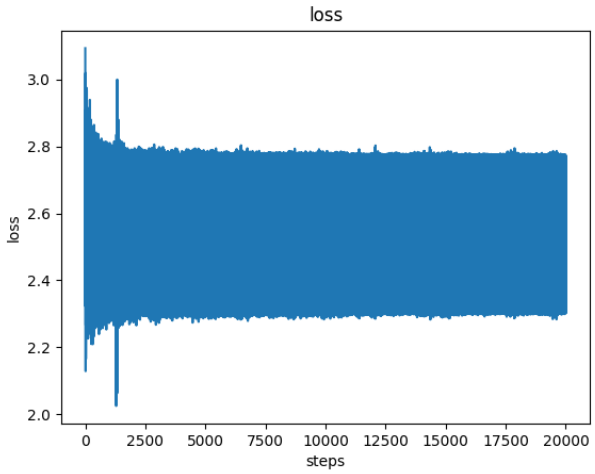


Figure A.80: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

bert 5 BB cosine

For the dataset BB, we used the Bi-directional bert encoder with 5 GNN layers and the bert model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

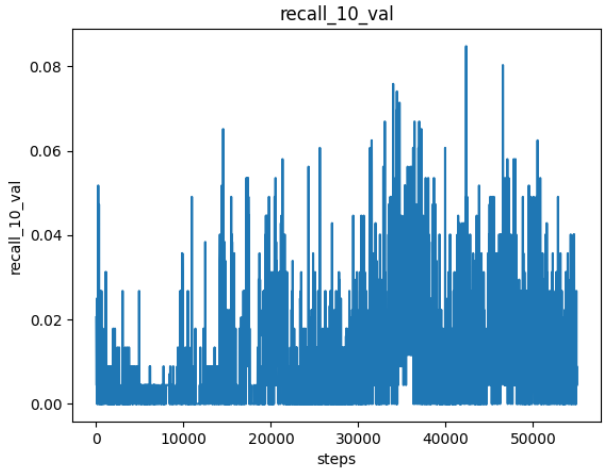


Figure A.81: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

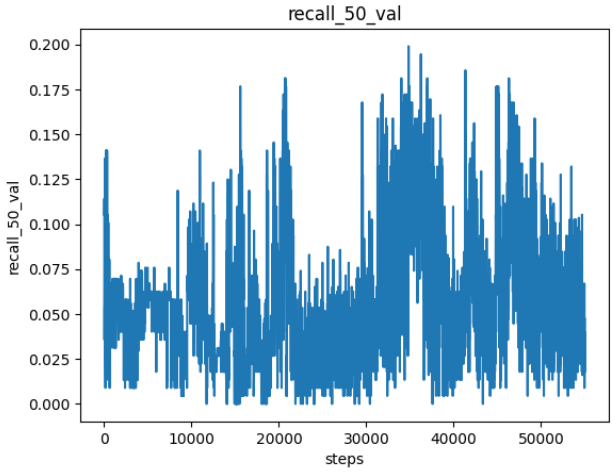


Figure A.82: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

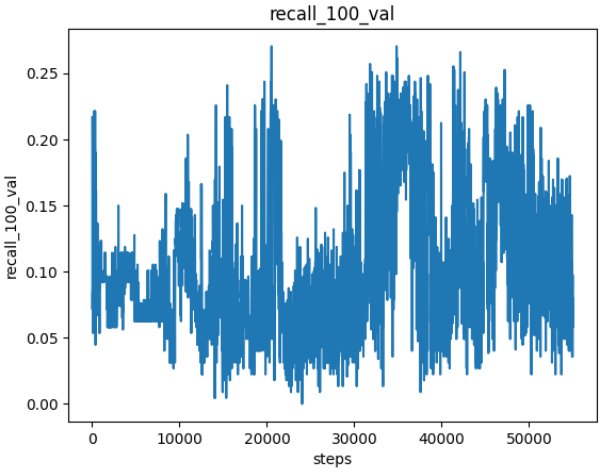


Figure A.83: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

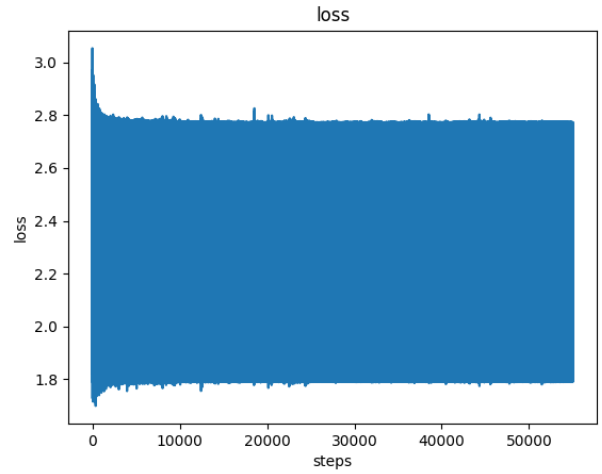


Figure A.84: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function cosine.*

bert 3 BB cosine

For the dataset BB, we used the Bi-directional bert encoder with 3 GNN layers and the bert model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

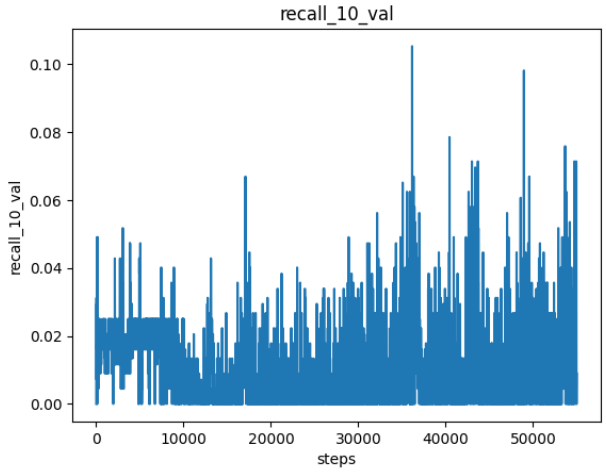


Figure A.85: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

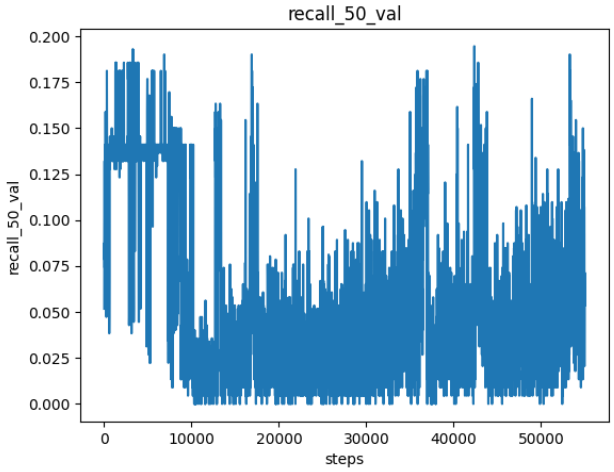


Figure A.86: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

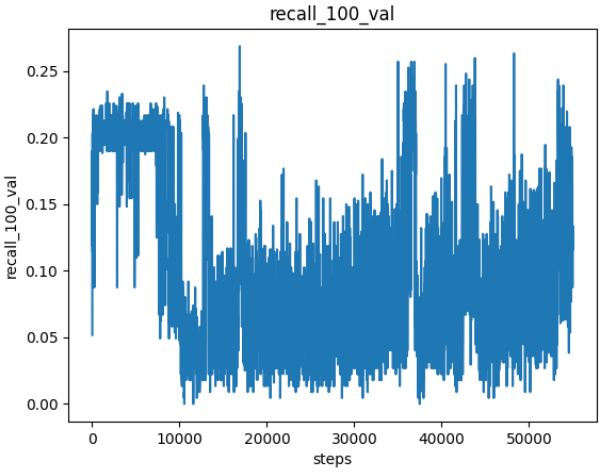


Figure A.87: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

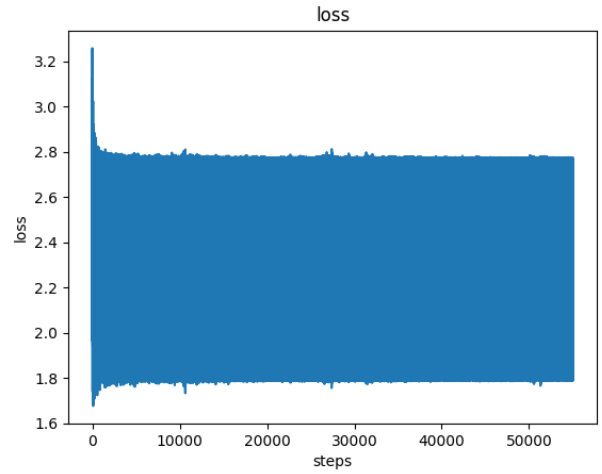


Figure A.88: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function cosine.*

bert 0 BB cosine

For the dataset BB, we used the Bi-directional bert encoder with 0 GNN layers and the bert model. The loss function used was cosine. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

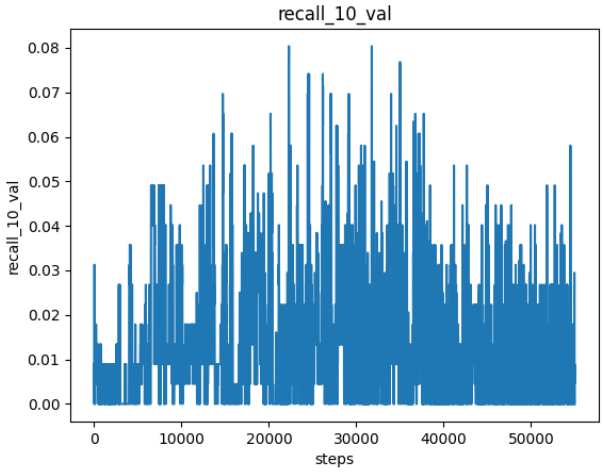


Figure A.89: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

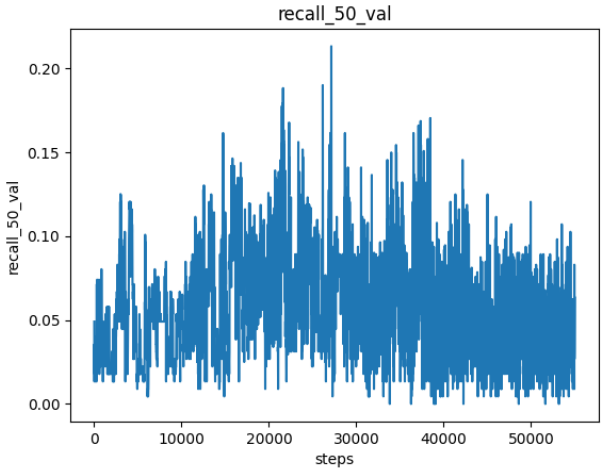


Figure A.90: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

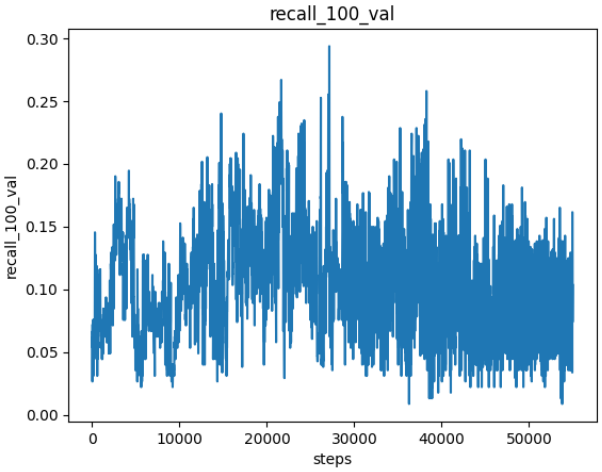


Figure A.91: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

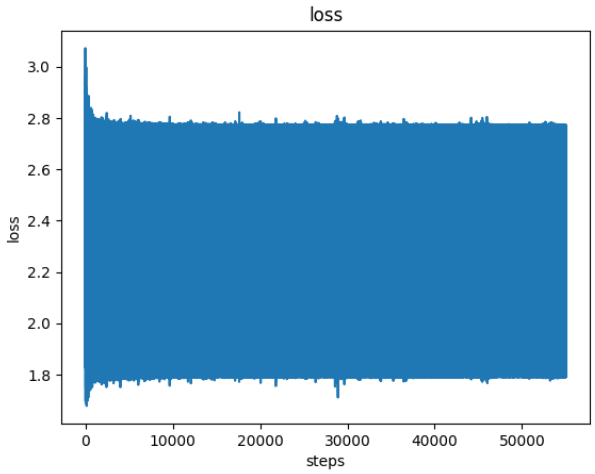


Figure A.92: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function cosine.*

bert 5 DBpeida Tripplet margin loss

For the dataset DBpeida, we used the Bi-directional bert encoder with 5 GNN layers and the bert model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

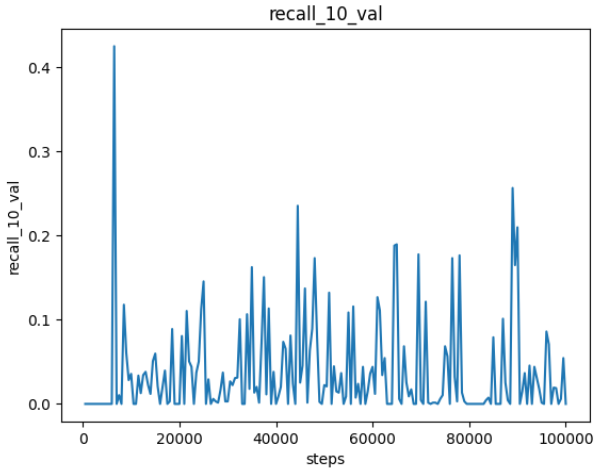


Figure A.93: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

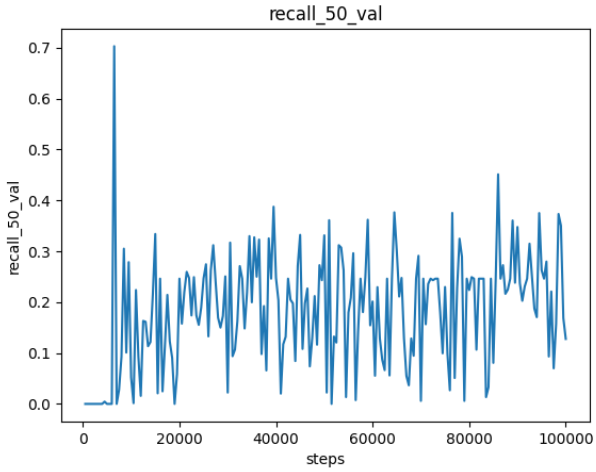


Figure A.94: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

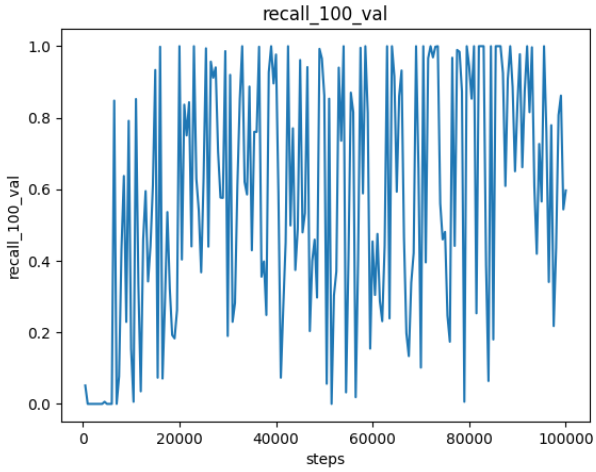


Figure A.95: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

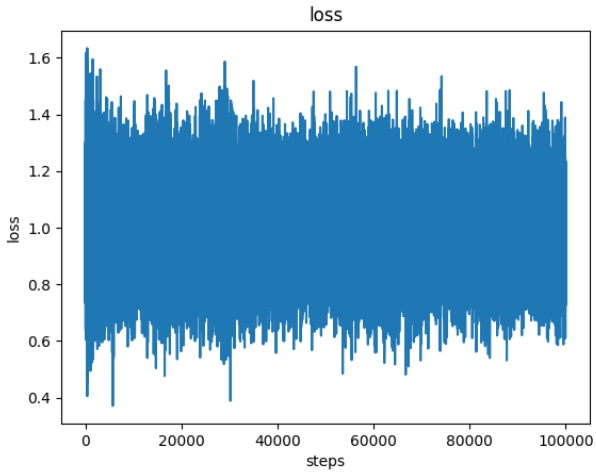


Figure A.96: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

bert 3 DBpeida Tripplet margin loss

For the dataset DBpeida, we used the Bi-directional bert encoder with 3 GNN layers and the bert model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

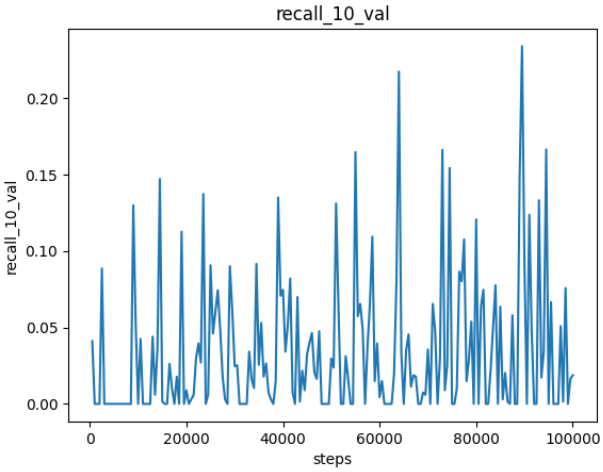


Figure A.97: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

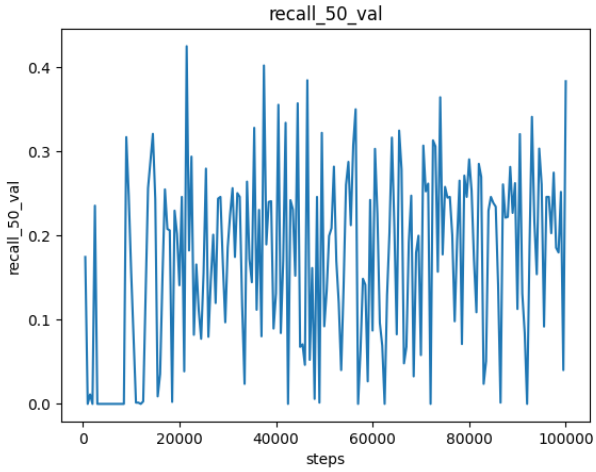


Figure A.98: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

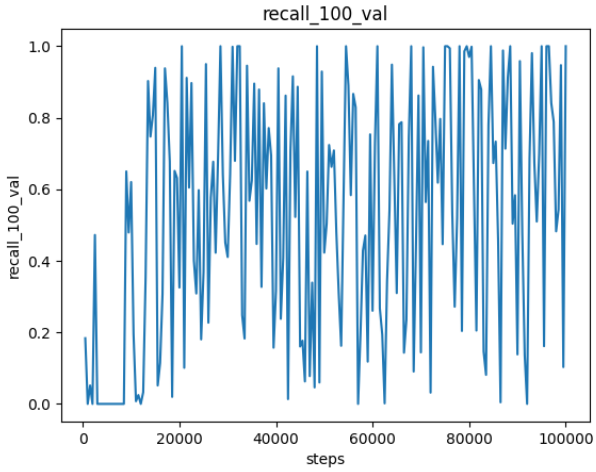


Figure A.99: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

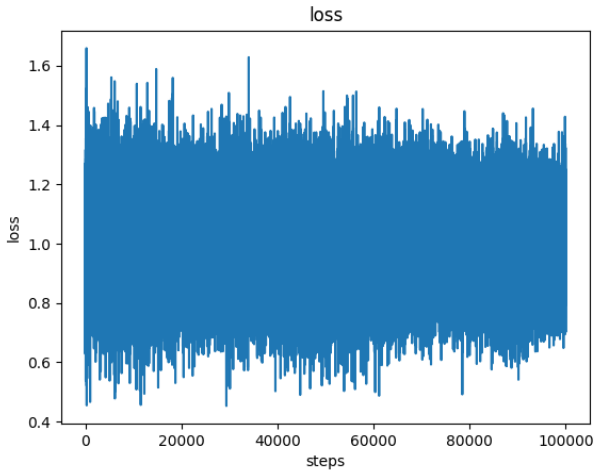


Figure A.100: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

bert 0 DBpeida Tripplet margin loss

For the dataset DBpeida, we used the Bi-directional bert encoder with 0 GNN layers and the bert model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

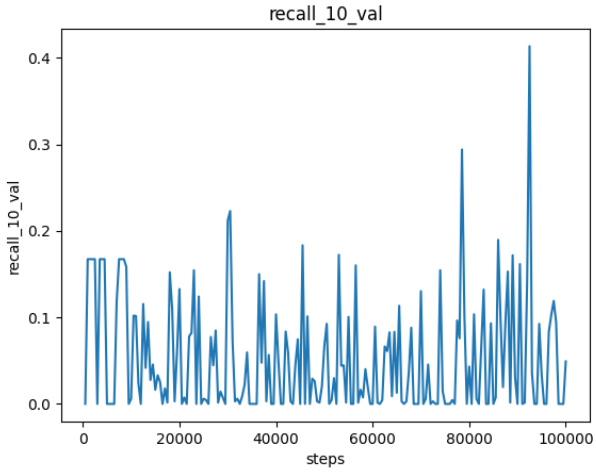


Figure A.101: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

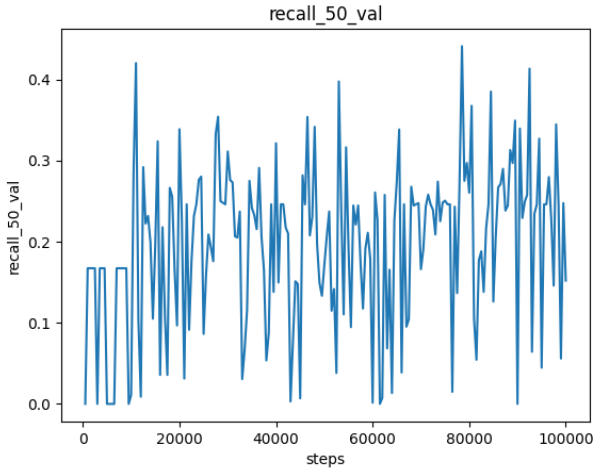


Figure A.102: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

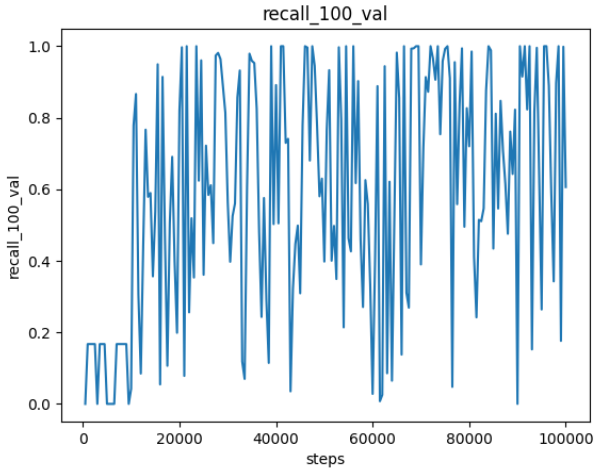


Figure A.103: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

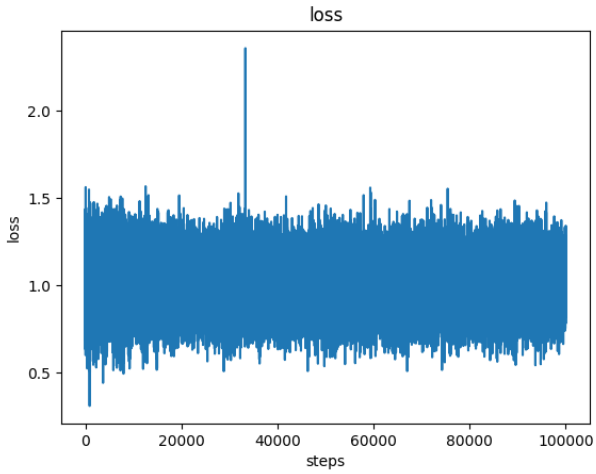


Figure A.104: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset DBpeida, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

bert 5 BB Tripplet margin loss

For the dataset BB, we used the Bi-directional bert encoder with 5 GNN layers and the bert model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

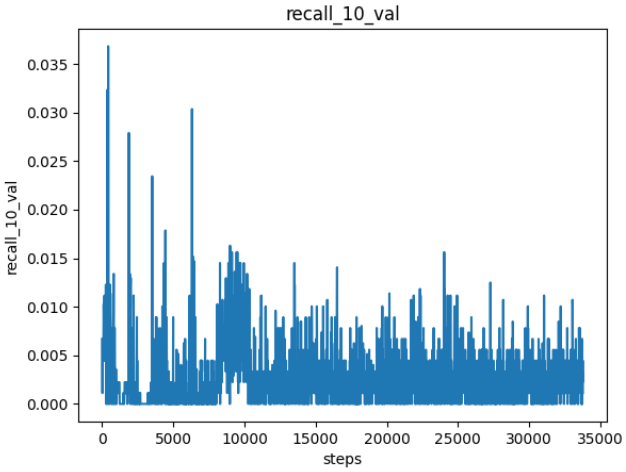


Figure A.105: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

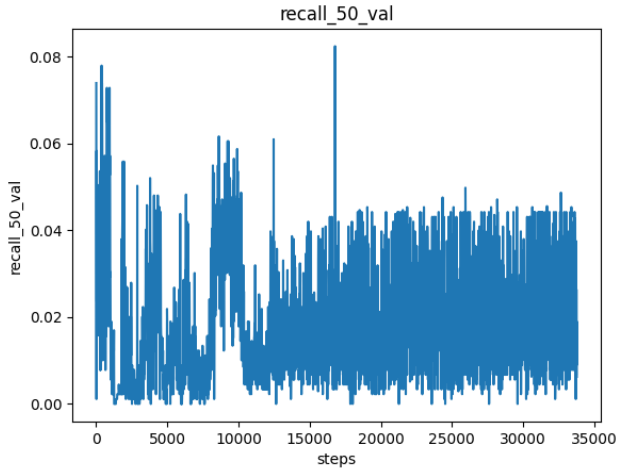


Figure A.106: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

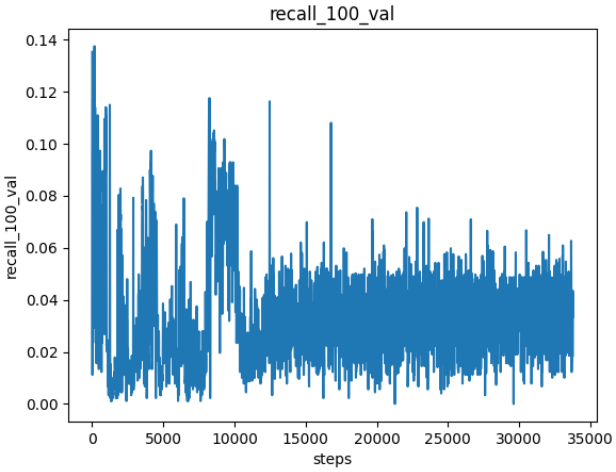


Figure A.107: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

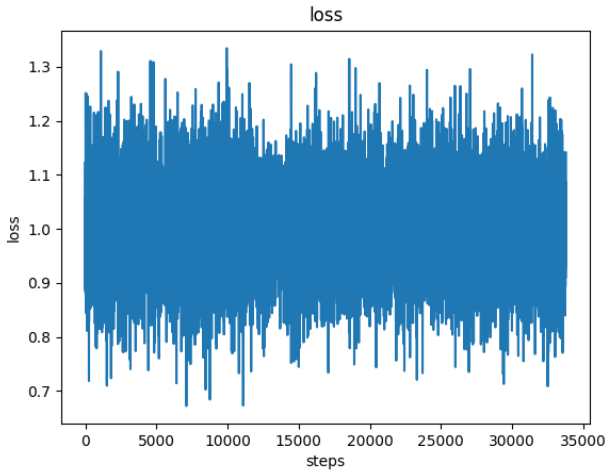


Figure A.108: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

bert 3 BB Tripplet margin loss

For the dataset BB, we used the Bi-directional bert encoder with 3 GNN layers and the bert model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

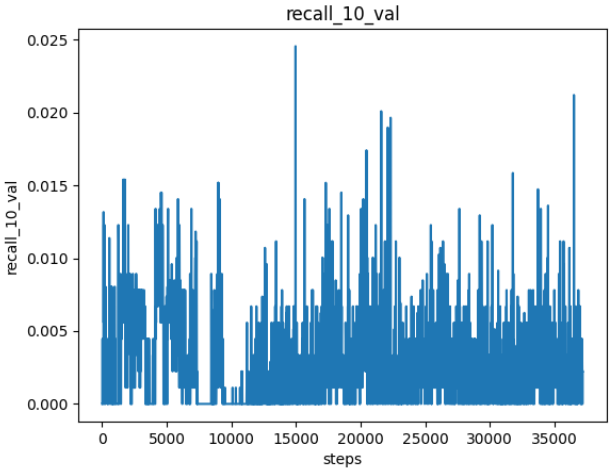


Figure A.109: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

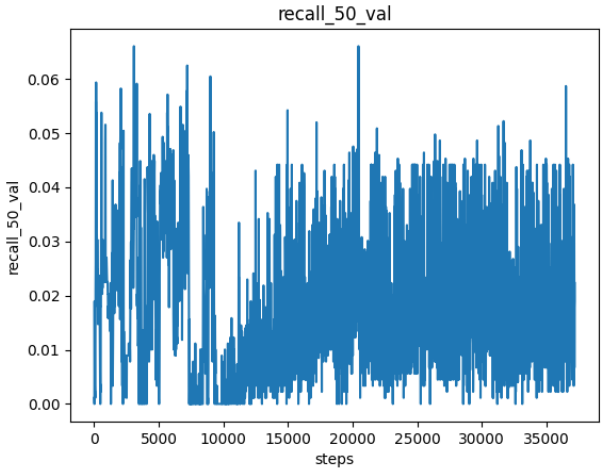


Figure A.110: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

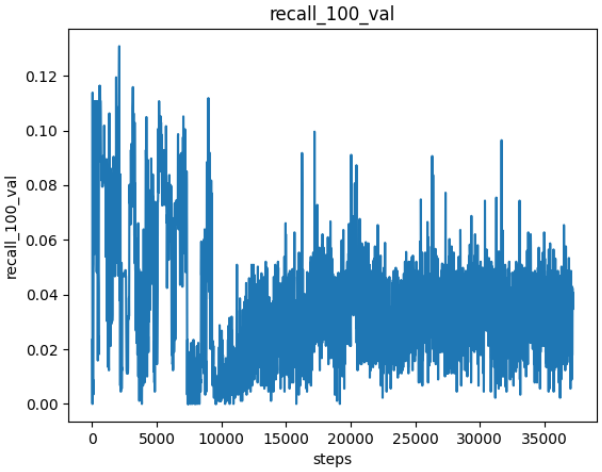


Figure A.111: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

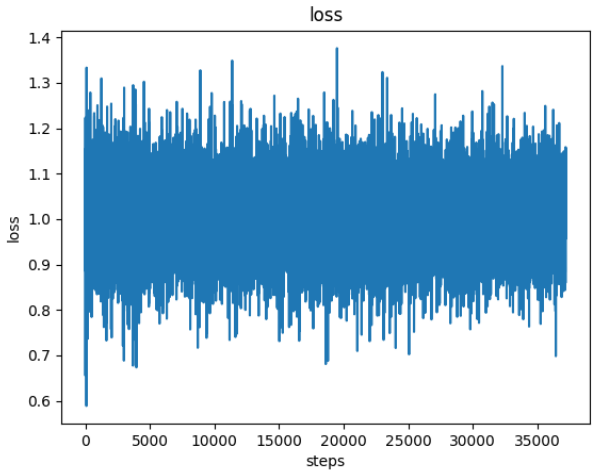


Figure A.112: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

bert 0 BB Tripplet margin loss

For the dataset BB, we used the Bi-directional bert encoder with 0 GNN layers and the bert model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

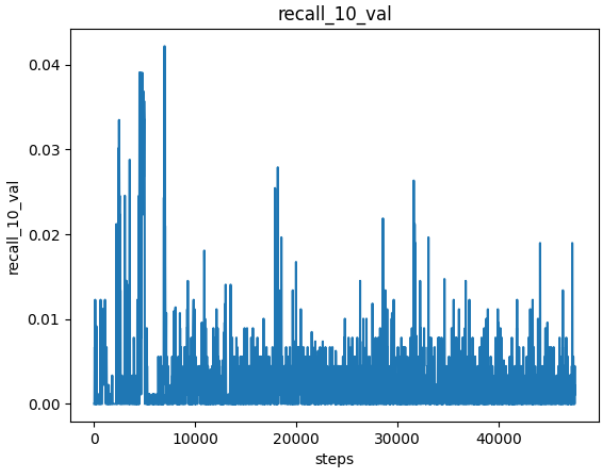


Figure A.113: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

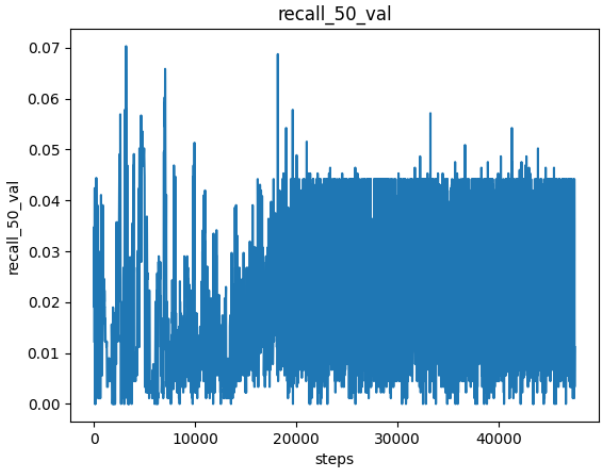


Figure A.114: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

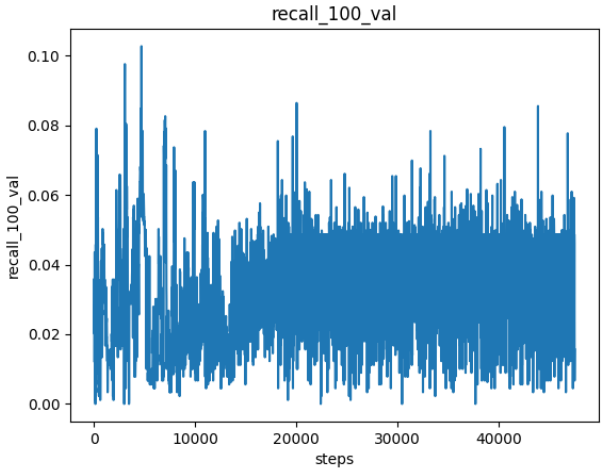


Figure A.115: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

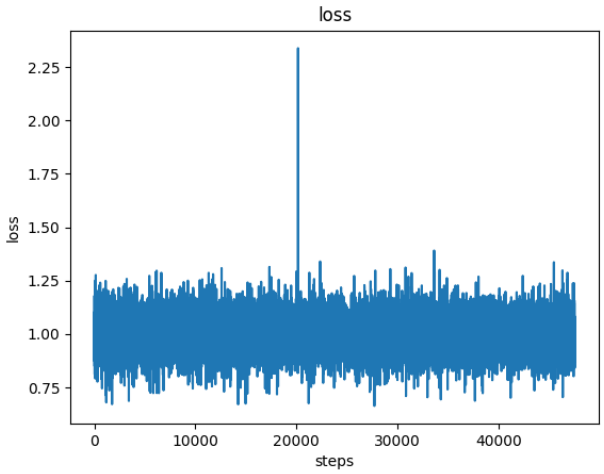


Figure A.116: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset BB, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

bert 5 Schema Tripplet margin loss

For the dataset Schema, we used the Bi-directional bert encoder with 5 GNN layers and the bert model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

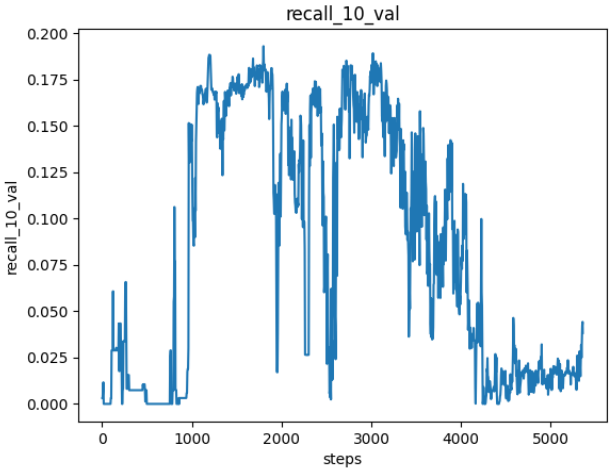


Figure A.117: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

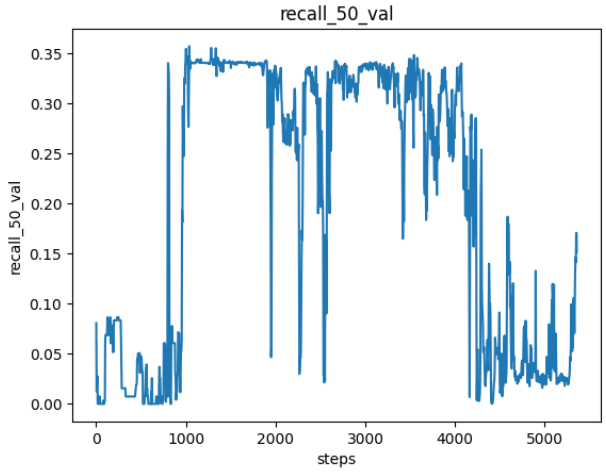


Figure A.118: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

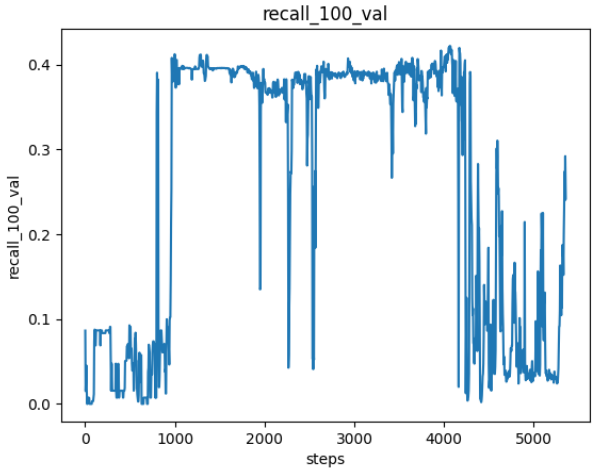


Figure A.119: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

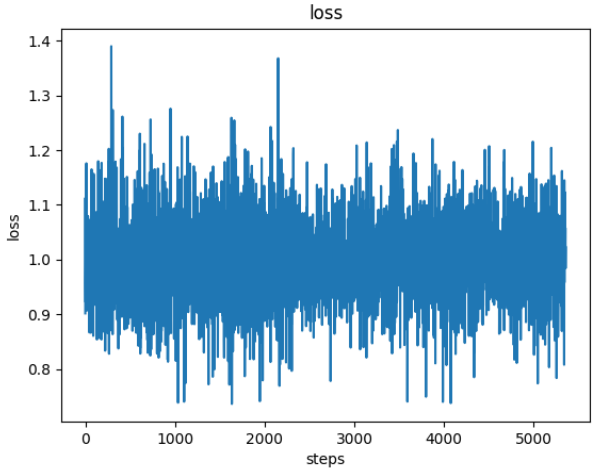


Figure A.120: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 5 GNN layers, and the loss function Tripplet margin loss.*

bert 3 Schema Tripplet margin loss

For the dataset Schema, we used the Bi-directional bert encoder with 3 GNN layers and the bert model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

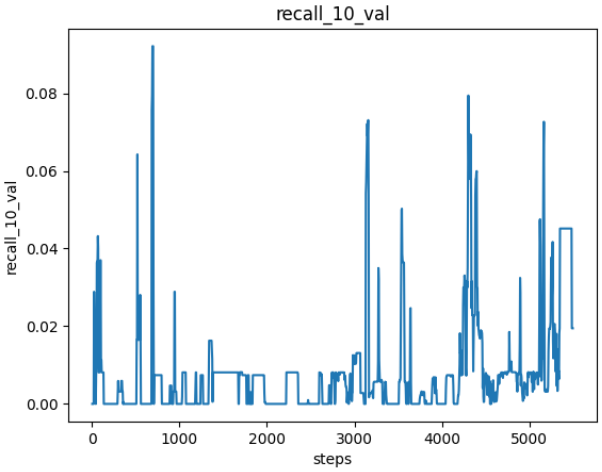


Figure A.121: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

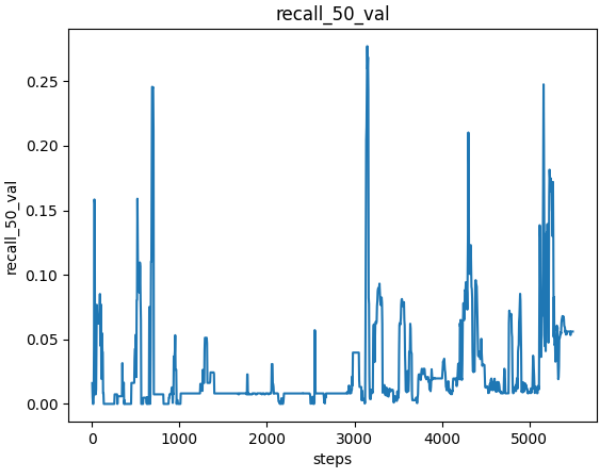


Figure A.122: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

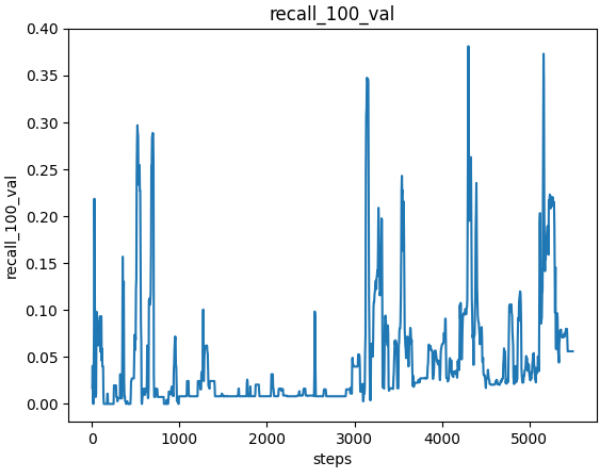


Figure A.123: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

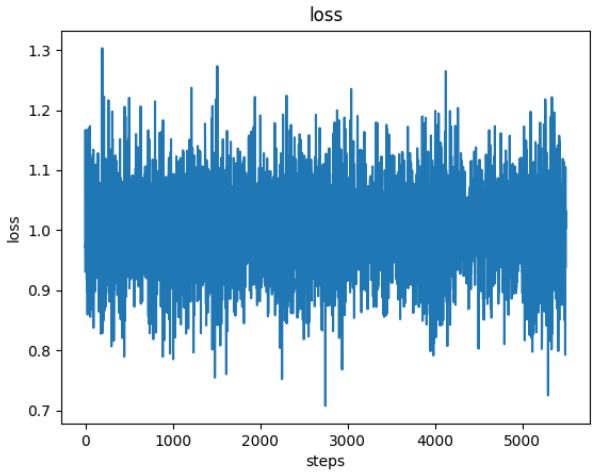


Figure A.124: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 3 GNN layers, and the loss function Tripplet margin loss.*

bert 0 Schema Tripplet margin loss

For the dataset Schema, we used the Bi-directional bert encoder with 0 GNN layers and the bert model. The loss function used was Tripplet margin loss. The following figures show the loss and recall values obtained during the training process. It is important to note that these values are indicative of the model's performance and can vary depending on the specific configuration and parameters used.

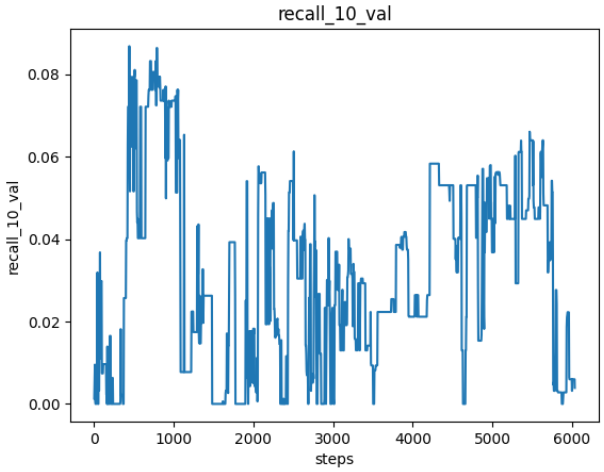


Figure A.125: *This image illustrates the recall@10 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

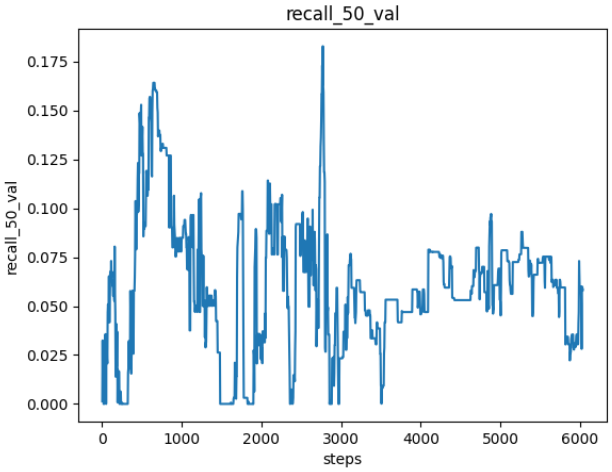


Figure A.126: *This image illustrates the recall@50 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

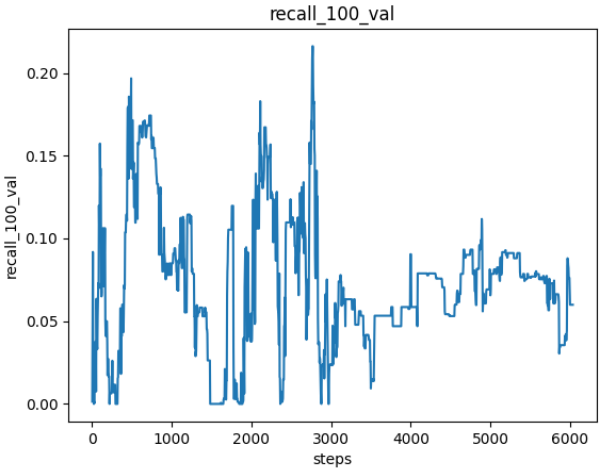


Figure A.127: *This image illustrates the recall@100 score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*

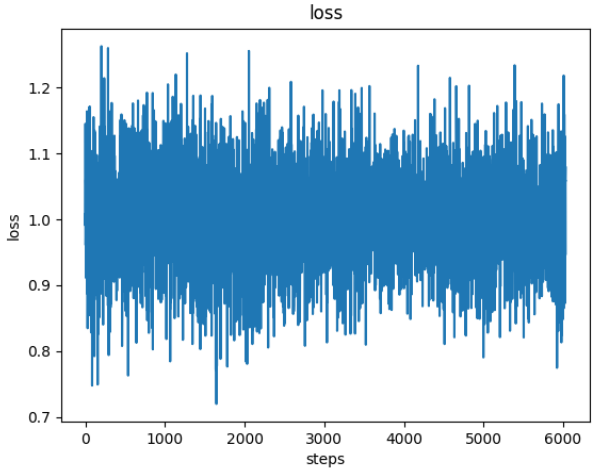


Figure A.128: *This image illustrates the loss score over time for different model configurations. The parameters for these configurations include the dataset Schema, the BERT model bert, the entity encoder Bi-directional bert encoder with 0 GNN layers, and the loss function Tripplet margin loss.*