People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University of 8 May 1945-Guelma-
Faculty of Mathematics, Computer Science and Science of Matter
Department of Computer Science

Master Thesis

Specialty: Computer Science

Option:

Computer systems

Theme

Dynamic QA System for Nosql Databases

Presented by:
Mihoubi Zakaria

Jury Members:

N	Full Name	Quality
1	Dr. KHEBIZI ALI	Chairman
2	Dr. AGGOUNE AICHA	Supervisor
3	Dr. BOUROUAIEH DOUADI	Examiner

June 2023.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to the Almighty Allah for granting me the strength, energy, determination, and resilience to successfully complete this humble endeavor.
I would like to extend my sincere appreciation to Dr. Aggoune Aicha, my esteemed supervisor, for her exceptional guidance, profound expertise, and invaluable advice throughout the entirety of this project. Her unwavering support and constructive feedback have played an instrumental role in the improvement and triumphant achievement of the objectives set forth in this work. Dr. Aggoune Aicha's dedication and commitment to my academic growth deserve special recognition, and I am truly grateful for her unwavering belief in my capabilities.
Furthermore, I would like to express my heartfelt gratitude to the esteemed members of the jury for their presence during the defense of my thesis and their meticulous examination of my research. I am profoundly appreciative of the time they have invested in carefully scrutinizing my work. Their insightful feedback, given during the presentation, holds immense value and will undoubtedly contribute to the further refinement of my research.
I would like to express my deep gratitude and heartfelt appreciation to my beloved family for their unwavering support, love, and encouragement throughout my academic journey.
To my beloved friends, I want to take a moment to express my heartfelt gratitude to each and every one of you. Your presence in my life has been an immeasurable gift, and I am truly thankful for the impact you have had on me. May the blessings of Allah shower upon each and every one of you, dear friends. The impact you have had on my life is immeasurable, and I am forever grateful. Regardless of where our individual paths may lead, I want you to know that I treasure the moments we have shared, and I harbor no regrets in crossing paths with each and every one of you.

ABSTRACT

Question-answering (QA) systems are crucial for retrieving information from large databases. While traditional systems focus on structured relational databases, the rise of NoSQL databases with a variety of query languages like MongoDB requires adapting these systems. In this study, we propose a NoT5QL (NoSQL T5) model for dynamic question answering of MongoDB based on fine-tuning of T5, a pre-trained transformer model for natural language processing (NLP) tasks. The developed dynamic question-answering system is specifically tailored for generating queries for MongoDB from natural language questions. To accomplish this task, a dataset called "MongoQpedia" (**Mongo**db **Q**uery **pedia**) was created using diverse questions from the Movies domain and annotated with MongoDB document-derived answers. The construction of MongoQpedia is based on data augmentation via paraphrasing, back translation, and named entity replacement techniques.

The evaluation of the NoT5QL model was performed through various metrics such as BLEU and ROUGE, comparing fine-tuned T5 small and base models. This provided insights into the impact of model size and complexity on MongoDB question-answering capabilities. The results of these experiments demonstrate the effectiveness of our approach in achieving high accuracy in answering questions related to MongoDB databases.

Keywords: *Question answering systems, NoSQL databases, T5, dataset construction, Data augmentation techniques, Fine-tuning.*

RÉSUMÉ

Les systèmes de question-réponse (QA) sont cruciaux pour récupérer des informations à partir de grandes bases de données. Alors que les systèmes traditionnels se concentrent sur les bases de données relationnelles structurées, l'émergence de bases de données NoSQL avec une variété de langages de requête tels que MongoDB nécessite d'adapter ces systèmes. Dans cette étude, nous proposons un modèle NoT5QL (NoSQL T5) pour la question-réponse dynamique de MongoDB basé sur l'affinage de T5, un modèle de transformation pré-entraîné pour le traitement automatique du langage naturel (TALN). Le système de question-réponse dynamique développé est spécifiquement conçu pour générer des requêtes pour MongoDB à partir de questions en langage naturel. Afin d'accomplir cette tâche, un ensemble de données appelé "MongoQpedia" (**Mongo**db **Q**uery **pedia**) a été créé en utilisant des questions diverses du domaine des films et annoté avec des réponses dérivées des documents MongoDB. La construction de MongoQpedia est basée sur l'augmentation des données via des techniques de paraphrase, de traduction inversée et de remplacement d'entités nommées.

L'évaluation du modèle NoT5QL a été réalisée à l'aide de différentes mesures telles que BLEU et ROUGE, en comparant les modèles T5 small et base affinés. Cela a permis d'obtenir des informations sur l'impact de la taille et de la complexité du modèle sur les capacités de question-réponse de MongoDB. Les résultats de ces expériences démontrent l'efficacité de notre approche pour atteindre une grande précision dans la réponse aux questions relatives aux bases de données MongoDB.

Mots-clés : *Systèmes de question-réponse, bases de données NoSQL, T5, construction d'ensemble de données, techniques d'augmentation des données, affinage.*

ملخص

تعد أنظمة الإجابة عن الأسئلة (QA) ضرورية لاسترجاع المعلومات من قواعد البيانات الكبيرة. في حين تركز الأنظمة التقليدية على قواعد البيانات العلائقية المنظمة، فإن ظهور قواعد بيانات NoSQL مع مجموعة متنوعة من لغات الاستعلام مثل MongoDB يتطلب تكييف هذه الأنظمة. في هذه الدراسة، نقترح نموذج NoT5QL (NoSQL T5) للإجابة عن الأسئلة الديناميكية ل MongoDB استنادا إلى الضبط الدقيق ل T5 ، وهو نموذج محول مدرب مسبقا لمهام معالجة اللغة الطبيعية (NLP). تم تصميم النظام الديناميكي للإجابة على الأسئلة خصيصا لتوليد استعلامات ل MongoDB من أسئلة اللغة الطبيعية. لانجاز هذه المهمة، تم إنشاء مجموعة بيانات تسمى MongoQpedia (**Mongo** db **Q**uery **pedia**) باستخدام أسئلة متنوعة من مجال الأفلام وتم توضيحها بالإجابات المشتقة من مستند MongoDB . يستند بناء MongoQpedia إلى زيادة البيانات من خلال إعادة الصياغة، والترجمة الخلفية، وتقنيات إستبدال الكيان المسمى.

تم إجراء تقييم نموذج NoT5QL من خلال مقاييس مختلفة مثل BLEU و ROUGE ، لمقارنة نماذج T5 الصغيرة والقعدية الدقيقة. قدم هذا رؤى حول تأثير حجم النموذج وتعقيده على قدرات إجابة أسئلة MongoDB . وتظهر نتائج هذه التجارب فعالية نهجنا في تحقيق دقة عالية

في الإجابة على الأسئلة المتعلقة بقواعد بيانات قاعدة بيانات MongoDB .

الكلمات المفتاحية: أنظمة إجابة الأسئلة، قواعد بيانات NoSQL ، T5 ، بناء مجموعة البيانات، تقنيات زيادة البيانات، الضبط الدقيق.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

GENERAL INTRODUCTION

The digital world is experiencing rapid growth and increasing complexity in terms of its volume (from terabytes to petabytes), variety (including structured, unstructured, and hybrid data), and velocity (with high-speed growth) [1] . The need to efficiently access and retrieve this vast volume of data has given rise to advanced techniques and technologies, including question-answering systems. These systems enable users to ask questions in everyday language and receive accurate answers, rather than providing a list of relevant documents like search engines typically do [2]. While question-answering systems have been extensively studied in the context of traditional relational databases, there has been a noticeable gap in the development of such systems for NoSQL databases, including MongoDB [3]. The existing works are based on deep learning and NLP (Natural Language Processing) techniques such as bag of words and parsing trees. However, these approaches are based on the external source, which is the knowledge graph in order to identify relevant entities and generate a list of natural language answers. Besides, some works are limited to some queries and required more details about the architecture and the dataset used. This thesis addresses these issues by focusing on question answering systems specifically designed for MongoDB using fine-tuned deep-learning models.

Due to the lack of benchmarks and datasets for deep learning of QA systems for MongoDB, the main contributions of this thesis lie in the creation of a new large-scale dataset, named MongoQpedia, and the proposition of NoT5QL model by fine-tuning of the T5 (Text-to-Text Transfer Transformer) model for the task of question answering in MongoDB.

To construct the MongoQpedia dataset, we first identified diverse and representative questions covering the domain of Movies using the Mongodb Movies Sample Database. These questions were then annotated with corresponding answers derived from MongoDB documents. In addition, to increase the diversity and quality of the dataset, we applied three data augmentation techniques: (1) Paraphrasing was used to generate alternative versions of the questions while preserving their original meaning, (2) Back translation was employed to translate the questions into a different language and then translate them back to English, capturing different phrasings and expressions, (3) Named entity replacement was used to replace movie-related named entities with similar entities, further expanding the dataset. By constructing this dataset, we aim to provide a valuable resource for training and evaluating question answering systems tailored to the unique characteristics of MongoDB's document-oriented structure.

The fine-tuning of the T5 model on the MongoQpedia dataset represents a significant advancement in the field of question-answering systems for NoSQL databases. By fine-tuning the T5 model on the MongoQpedia dataset, we enhance its ability to understand and generate accurate responses to questions posed in natural language within the context of MongoDB's document-oriented data model.

It is worth noting that this work represents one of the first attempts to address the lack of development in question answering systems for NoSQL databases, particularly MongoDB. While question answering systems have thrived in the realm of traditional relational databases, the unique characteristics and data model of NoSQL databases present novel challenges and opportunities. Our research fills this void by focusing specifically on MongoDB and leveraging deep learning techniques to enable effective question-answering.

To summarize, the objectives of this thesis are twofold. Firstly, we aim to create the

MongoQpedia dataset, consisting of movie-related questions and their corresponding MongoDB query-based answers. This dataset serves as a benchmark for evaluating the performance of question answering systems in MongoDB. Secondly, we fine-tune the T5 model on the MongoQpedia dataset to develop a question answering system tailored to MongoDB. Through extensive experimentation and evaluation, we aim to demonstrate the effectiveness and practicality of our approach. The findings of this thesis have far-reaching implications for the field of NoSQL databases and question answering systems.

This report is divided into four chapters:

1. **Chapter One: Querying NoSQL database**

   This chapter will explore NoSQL databases, covering their various subcategories and principles. We will discuss the different kinds of NoSQL databases, such as key-value stores, document databases, wide-column stores, and graph databases. Additionally, we will delve into important topics and examine the query languages employed in NoSQL databases. By the end of this chapter, you will have a comprehensive comprehension of NoSQL databases and their role in handling and analyzing vast amounts of data.

2. **Chapter Two: An overview of QA systems for NoSQL databases**

   The objective of this chapter is to provide an overview of QA (Question-Answering) systems, focusing on a bibliographic study of deep learning techniques used in QA systems like BERT, GPT, and others. We will specifically explore the application of QA systems for NoSQL querying, with a particular emphasis on document-oriented databases such as MongoDB.

3. **Chapter Three: Deep learning-based QA for MongoDB database**

In this chapter, we offer an extensive and comprehensive contribution. Firstly, we present a detailed description of the process involved in creating the MongoQpedia dataset. We discuss the various techniques and algorithms used for data augmentation to enhance the quality and diversity of the dataset. Additionally, we provide an in-depth explanation of the conceptual architecture of our NoT5QL model. This encompasses the design and structure of our model specifically tailored for tackling NoSQL querying challenges. By covering these aspects, we aim to provide a thorough understanding of the steps involved in dataset creation, augmentation techniques, and the architecture of our innovative NoT5QL model.

4. **Chapter Four: Implementation and Experimentation**

   This chapter will examine the specific aspects of implementation and the assessment of performance. The chapter is structured into three primary sections. The initial section concentrates on the development environment, encompassing the necessary hardware and software prerequisites for establishing the system. The subsequent section presents a summary of the libraries employed during the implementation procedure. Lastly, the third section elucidates the principal components of the implementation and furnishes an intricate analysis of the outcomes acquired from our Q&A system. At the conclusion of this chapter, readers will possess a comprehensive comprehension of the development procedure and the diverse tools employed in implementing our Q&A system.

CHAPTER 1 QUERYING NOSQL DATABASE

1.1 Introduction

The amount of data that needs to be kept and organized has increased significantly as a result of the Internet's recent rapid growth [4]. Traditional relational databases, which were created for modestly organized data collections, are now difficult to use. These databases follow a strict schema, which is a preset organization of data in tables, and require a fixed structure before they can be made. This rigid approach has problems scaling and handling massive amounts of heterogeneous data, which are more common in today's internet-centric environment [4].
NoSQL data stores have become a substitute solution to get around these restrictions. NoSQL databases, in contrast to relational databases, offer flexible schema that enable the storage of enormous quantities of heterogeneous data without a predefined structure.As a result, NoSQL data stores are well suited for managing massive amounts of data and have shown encouraging results for horizontal scalability [5].
In this chapter, we will examine NoSQL databases, along with the numerous subcategories and guiding principles that surround them. We will talk about the different

types of NoSQL databases, including key-value stores, document databases, wide-column stores, and graph databases. We'll also look at some of the key issues in this area and the various query languages used for NoSQL databases, giving a thorough understanding of NoSQL databases and their function in managing and analyzing enormous volumes of data by the end of this chapter.

1.2 What is NoSQL?

SQL databases struggle to handle an excessive amount of complex or partially structured data, fast-paced processing, and expanding horizontally. NoSQL databases offer better speed, versatility in terms of scaling, and more adaptable schema design, which are benefits over SQL databases in these scenarios. This shift towards using NoSQL databases has arisen due to the recognition of the limitations of SQL in specific situations [6].

In 1998, Carlo Strozzi coined the term NoSQL to describe a new approach for managing a large amount of heterogeneous data [7]. NoSQL, which stands for "Not Only SQL," is a type of data storage alternative to traditional databases, designed to handle large amounts of data, like the constantly growing data on Facebook [6].

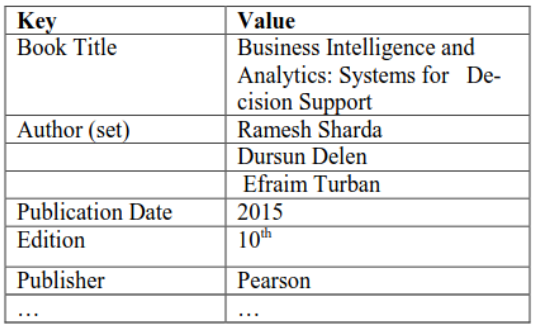
NoSQL is a non-relational database management system, known for quick information retrieval and portability. It evolved from the traditional relational database (RDB) system and typically interacts with the UNIX operating system [8]. NoSQL databases are non-relational, open-source, distributed, and horizontally scalable for high performance. They do not organize data in related tables like relational databases, and instead store data in a non-normalized manner. Being open-source, the code is publicly available for inspection, modification, and recompilation. The distributed nature means data is stored across multiple machines and managed through data replication [9].

1.3 Categories of NoSQL datastores

Four main categories of NoSQL databases have developed over time: key-value database, wide-column stores, document databases, and graph databases [6].

1.3.1 Key-value databases

Key-value data stores are basic in nature, but are highly efficient and powerful. They feature a simple application programming interface (API) and allow users to store data in a flexible manner, without having to follow a strict schema. The data is stored as a "key-value" pair, where a string (key) represents the reference to the actual data (value). These stores operate similarly to hash tables and are quicker than RDBMS due to their use of keys as indexes.
The key-value data model is simple, with data stored in a dictionary-like format that can be accessed using the specified key. Modern key-value data stores prioritize scalability over consistency, meaning that advanced querying and analytics functions like joins and aggregate operations are omitted. Despite this, key-value stores offer high concurrency, fast lookups, and the ability to store large amounts of data. One weakness of this data model is the lack of a defined schema, making it harder to create custom views of the data [5].
Some notable DBMS providers using key-value data stores are Amazon DynamoDB, RIAK, Redis.



Key	Value
Book Title	Business Intelligence and Analytics: Systems for Decision Support
Author (set)	Ramesh Sharda
	Dursun Delen
	Efraim Turban
Publication Date	2015
Edition	10th
Publisher	Pearson
…	…

FIGURE 1.1: An example of how key-value pairs are stored in a NoSQL database [4].

1.3.2 Column-Oriented databases

Column stores in NoSQL are a blend of row and column storage, not just pure columnar databases like in traditional relational systems (see Fig 1.2). Despite borrowing the concept of column-based storage from columnar databases and adding columnar capabilities to row-based databases, column stores do not store data in tables, instead opting for a massively distributed architecture[5].
Column family stores enable searches using range queries and regular expressions on indexed values or row keys, reducing the impact of the database operations to only the relevant keys related to the query, rather than the entire row from disk. In addition to typical accessors and mutators, column-oriented databases commonly support operations such as OR, IN, and AND. A well-known example is the Cassandra API, which offers three main functions: get, insert, and delete, with parameters of table, key, and columnName [10].

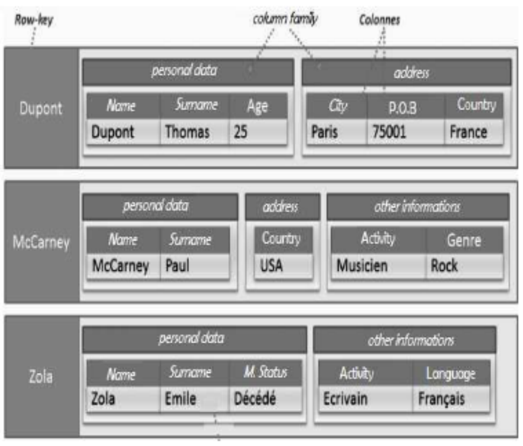
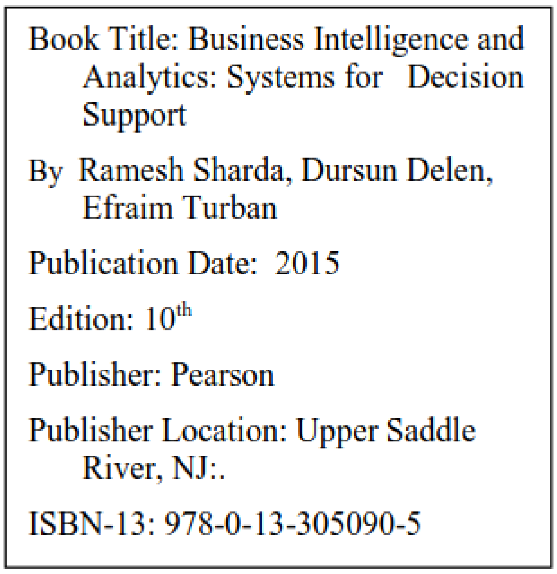


FIGURE 1.2: Example of the records storage example in a column-oriented database [11]

1.3.3 Document-oriented databases

Document-oriented databases store data in the form of documents, offering high performance and the ability to scale horizontally. The documents within a document-oriented database resemble records in relational databases, but with more flexibility as they do not require a specific schema. The documents are in standard formats such as XML, PDF, JSON, etc. Unlike relational databases, where records in the same database have the same data fields with empty fields for unused data, each document in a document store may have similar or dissimilar data. The documents are identified by a unique key, which can be a simple string or a reference to a URI or path.
Document stores are slightly more complex than key-value stores as they allow for the key-value pairs to be encapsulated within a document, also known as key-document pairs [5] .
A document-oriented type of NoSQL database is selected for a question-answering system because of its flexibility in storing unstructured or semi-structured data, scalability, and speed when it comes to information retrieval operations. The schema-less nature of document stores also allows for easy integration and management of changing data structures, making it a suitable choice for a question-answering system.

Fig. 1.3 gives an example of document-oriented datastore.


Book Title: Business Intelligence and Analytics: Systems for Decision Support

By Ramesh Sharda, Dursun Delen, Efraim Turban

Publication Date: 2015

Edition: 10th

Publisher: Pearson

Publisher Location: Upper Saddle River, NJ:.

ISBN-13: 978-0-13-305090-5

FIGURE 1.3: An example of a book stored in a document-oriented NoSQL database [4]

In this thesis, we focused on MongoDB as an example of NoSQL database, since it is the most popular NoSQL database and includes JSON-and BSON-based documents for storing data [12].

1.3.4 Graph databases

Graph databases store information in a graph structure, where nodes represent objects and edges represent the relationships between these objects. These nodes can have properties associated with them. The databases use a method called "index-free adjacency," where each node directly points to the adjacent node, allowing for the efficient traversal of millions of records. The focus of these databases is on the connections between data and they can handle semi-structured data with ease, providing schema-less storage.
Queries in graph databases are expressed as traversals, making them faster than traditional relational databases. They are also scalable and user-friendly while being

ACID-compliant and offering rollback support [5].

Applications that require managing data with numerous relationships are more appropriate for graph databases. These databases can replace time-consuming operations as recursive joins with more efficient methods like graph traversal and graph pattern matching. Graph traversal starts with one node and then follows the description of the query to traverse to other nodes. The graph pattern matching technique tries to find a specific pattern in the original graph.

In Neo4j graph DBMS, each vertex and edge in the graph has a "mini-index" of connected objects, so the size of the graph doesn't affect its performance during a traversal and the cost of a single step remains constant. There is also a global adjacency index, but it is only used to locate the starting point of a traversal. Indexes are crucial for quickly retrieving vertices based on their values and serve as the starting point for a traversal [10].

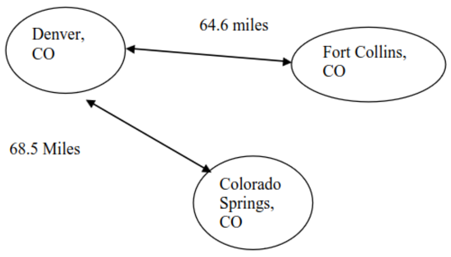


FIGURE 1.4: An example of a Graph NoSQL Database for the distance between cities[4] .

1.4 CAP and BASE

NoSQL databases are made to meet the scalability and performance needs of web-based applications that can't be met by conventional relational databases. These databases are referred to as "NoSQL" databases since they integrate many more characteristics in addition to the features of traditional databases. This is due to their contrast in priorities and architecture to typical relational databases utilizing SQL. While NoSQL databases adhere to BASE (Basically Available, Soft State, Eventual consistency) principles or CAP (Consistency, Availability, Partition Tolerance), relational databases strictly adhere to the ACID (Atomicity, Consistency, Isolation, and Durability) properties [13].

1.4.1 CAP theory

Professor Eric Brewer proposed the well-known CAP theorem in the year 2000 [14]. The CAP theorem stands for Consistency, Availability, and Partition Tolerance. The fundamental idea of the CAP theorem is that a distributed system can only satisfy two of the three district demands at once [15] .

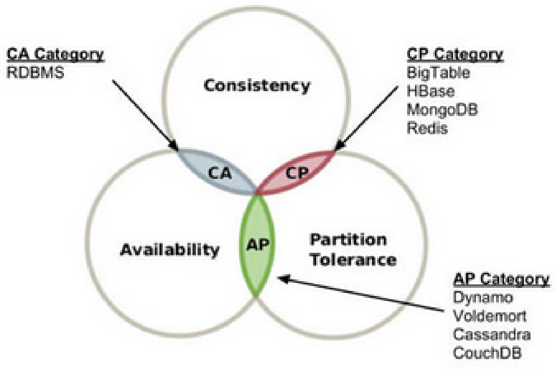


FIGURE 1.5: CAP Theorem [16]

This theorem relies on three properties [17]:

1. **Consistency:** The first component of CAP. A response that is appropriate to the desired service specification is what is meant by consistency. Informally speaking, consistency is the property that each server gives the correct response to each request.

2. **Availability:** The second component of CAP. The quality of having a response for every request at some point. Although it is obvious that a quick response is better than a slow response. The theorem shows that even the requirement of a response is enough to cause issues (In most real systems, of course, a late answer is just as terrible as none at all.).

3. **Partition tolerance:** The Third component of CAP. Partition tolerance, in contrast to the other two requirements, is more of a statement about the underlying system than it is about the service itself: communication between the servers is unreliable, and the servers can be divided into multiple groups that are unable to communicate with one another. We represent a partition-prone system as a communication-flawed system, where communications can be lost permanently or delayed indefinitely (Again, practically speaking, a long-delayed message may as well not be delivered.).

1.4.2 BASE properties

The term "BASE" was deliberately selected to differentiate from the ACID paradigm, with a focus on "Availability" and "Performance". Creating a database that follows the ACID principles can be challenging, so consistency and isolation are often compromised, leading to the widespread adoption of the BASE approach. The fundamental idea behind BASE is that data consistency is the responsibility of the developer, not the database [13].
The BASE properties are [18]:

1. **Basically-Available:** A distributed system must be able to acknowledge any incoming requests, even if they result in a failure message.

2. **Soft-state:**As new information is received, the system may continue to change its states.

3. **Eventually-consistent:** The system's parts might not always display the same value or state of a record. But eventually, they will resolve it with the passage of time.

1.5 NoSQL query languages

Query languages are useful for accessing relevant data from data storage. SQL query standards are being established, and many categories are being offered . NoSQL queries do not have a universally accepted industry standard [3]. The purpose of a Query language is to allow us to perform CRUD operations (Create, Read, Update, and Delete) on databases.
CRUD refers to the four fundamental functions that a software program must be capable of executing: Create, Read, Update, and Delete. This requires users to be able to generate data, access the information in the user interface through reading it, modify or change the data, and erase the data if necessary [19]. We give in the following part a definition of each operation [20]:

1. **Create Operation:**
   In a database, the CREATE function adds a fresh item. A row in a database table is referred to as a record in a relational database management system (RDBMS), and the columns are referred to as attributes or fields. The CREATE operation updates the table with one or more new records that have distinct field values. The same idea holds true for NoSQL databases. A new document with its attributes (for instance, a JSON-formatted document) is added to the collection, which is comparable to a table in an RDBMS, if the NoSQL database employs a document-oriented approach. The CREATE action adds an item, which is equivalent to a record, to the table in NoSQL databases like DynamoDB.

2. **Read Operation:**
   Based on a set of search parameters, the READ action obtains information from a database table, collection, or bucket. All records as well as some or all of the fields within them can be retrieved using the READ function.

3. **Update Operation:**

The UPDATE function is used to alter existing records in the database. For instance, it could be used to update a customer's address or change the price of a product in a product database. Like the READ operation, the UPDATE function can modify records across the entire database or just a select few based on specific criteria. The UPDATE operation can change and persist alterations to a single field or multiple fields within the record. If multiple fields need to be updated, the database system makes sure that all changes are made or none at all. Some big data systems do not support the UPDATE operation and instead only permit a timestamped CREATE operation, which creates a new version of the row each time it is executed.

4. **Delete Operation:**
   The DELETE operation enables users to eliminate records from the database. A hard delete completely removes the record, while a soft delete marks the record but keeps it in place. This is useful in situations such as payroll, where records of former employees need to be kept even after they have left the company.

We present in the rest of this section the principal query languages for NoSQL data stores.

1.5.1 MongoDB Query Language (MQL)

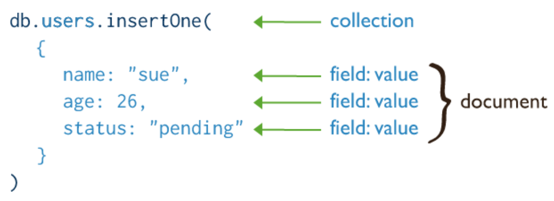
MongoDB offers a JSON-based query language that consists of two main components [21]:

1. The "find" query retrieves documents that meet a specific set of conditions and takes a query and a projection parameter. It returns a cursor to the matching documents, and optional modifiers can be added to impose limits and sort orders.

2. The "aggregate" query, on the other hand, enables the creation of processing pipelines, where each document in a collection passes through multiple stages to perform more complex computations.

A number of users are interested in transitioning from SQL to NoSQL data stores due to the efficient handling of unstructured data by NoSQL data stores. [22] .
MongoDB Query syntax for the CRUD operations is presented as follows:

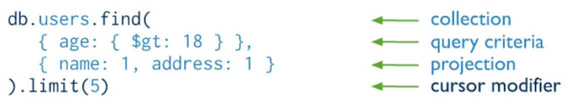
- **Create:** To insert new records into a MongoDB collection, one can utilize functions and BSON objects. "Insert" is the function provided for this purpose. Upon inserting a new object into a MongoDB collection, a unique field named "_id" is automatically added to the object, serving as an index [23]. The following example demonstrates how to add a new user to the "users" collection in MongoDB using the MongoDB shell:



```
db.users.insertOne(
  {
    name: "sue",
    age: 26,
    status: "pending"
  }
)
```

FIGURE 1.6: Inserting one document to the "users" collection [24]

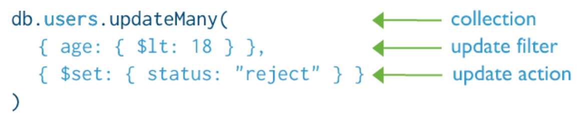
- **Read:** In MongoDB, you can retrieve data by using the "find" function. Unlike traditional relational databases, MongoDB does not have tables; instead, it stores data in collections that are flexible in structure. The items within these collections can have different fields [23]. When using the "find" function in MongoDB, you can specify query filters or conditions that determine which documents should be returned[24]. The example below shows how to read the specific document:



```
db.users.find(
   { age: { $gt: 18 } },
   { name: 1, address: 1 }
).limit(5)
```

FIGURE 1.7: Find all the documents in the "users" collection that satisfy the filter [24]

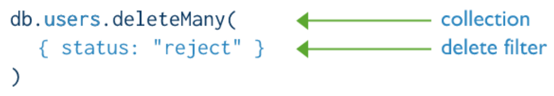
- **Update:** In MongoDB, updating records, whether it's just one or many, is made easy [23]. You can specify criteria, also known as filters, to select the documents to be updated. The syntax for these filters is the same as that used for read operations [24].



```
db.users.updateMany(
   { age: { $lt: 18 } },
   { $set: { status: "reject" } }
)
```

FIGURE 1.8: Update all the documents in the "users" collection that satisfy the filter [24]

- **Delete:** In MongoDB, the function for deleting data from a collection is "remove" [23]. As with updating records, you can specify criteria, or filters, to determine which documents should be deleted. The syntax for these filters is consistent with that used in read operations [24].



```
db.users.deleteMany(
   { status: "reject" }
)
```

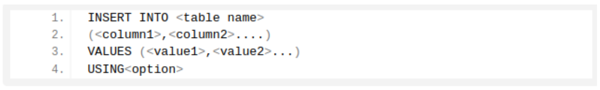
FIGURE 1.9: Delete all the documents in the "users" collection that satisfy the filter [24]

MongoDB would be the ideal choice for making a question-answering system because of its document-oriented design that supports unstructured or semi-structured data, its ability to scale with large volumes of data, and its capability to manage complex and dynamic data structures. Furthermore, MongoDB has strong community support and a wide set of tools and services. Its indexing and querying features also make it perfectly suited for systems that require fast information retrieval and the ability to retrieve data based on multiple conditions. For these reasons, the use of MongoDB for the query-answering systems is very suitable to give the opportunity to easily use NoSQL data via natural language.

1.5.2 The Cassandra Query Language (CQL):

This query language is used in the Cassandra database. It is also designed to store data in column-oriented Databases. In CQL (Cassandra Query Langage), columns are the fundamental data structures, and they can handle both structured and unstructured data. Furthermore, Key spaces are used as a database in CQL [25].
CQL provides a paradigm similar to SQL in that data is stored in tables with rows of fields.

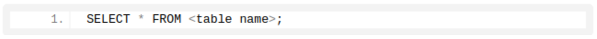
- **Create:**



```
INSERT INTO <table name>
(<column1>,<column2>....)
VALUES (<value1>,<value2>...)
USING<option>
```

FIGURE 1.10: Create operation for CQL [26]

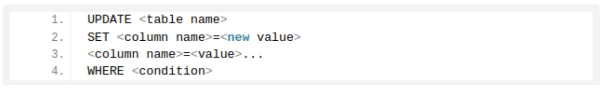
- **Read:**



```
SELECT * FROM <table name>;
```

FIGURE 1.11: Read operation for CQL [26]

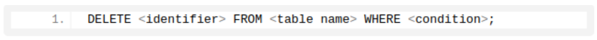
- **Update:**



```
UPDATE <table name>
SET <column name>=<new value>
<column name>=<value>...
WHERE <condition>
```

FIGURE 1.12: Update operation for CQL [26]

- **Delete:**



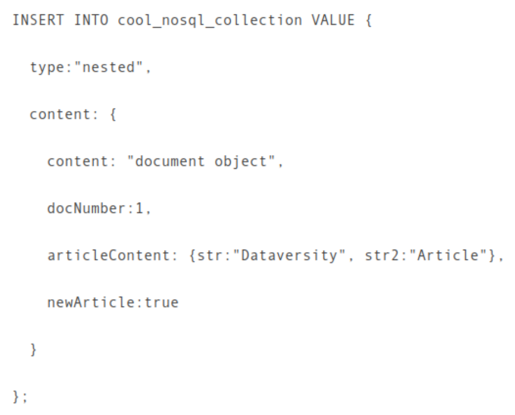
```
DELETE <identifier> FROM <table name> WHERE <condition>;
```

FIGURE 1.13: Delete operation for CQL [26]

1.5.3 UnQL

This language is specifically designed for CouchDB a document-oriented database. Unstructured Query Language (UnQL) uses structural recursion to extract information from semi-structured data. UnQL, unlike to MongoDB, offers a model similar to SQL in that data is stored in documents.

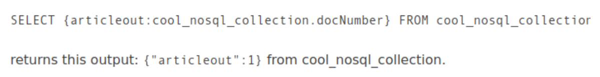
- **Create:**:



```
INSERT INTO cool_nosql_collection VALUE {
  type:"nested",
  content: {
    content: "document object",
    docNumber:1,
    articleContent: {str:"Dataversity", str2:"Article"},
    newArticle:true
  }
};
```

FIGURE 1.14: Create operation for UnQL [27]

- **Read:** UnQL has a straightforward syntax for selecting and filtering data, which includes the use of "select" and "where" statements with pattern matching capabilities [28].

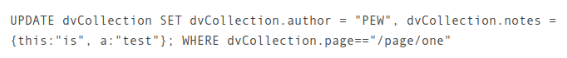


```
SELECT {articleout:cool_nosql_collection.docNumber} FROM cool_nosql_collectior
```

returns this output: `{"articleout":1}` from cool_nosql_collection.

FIGURE 1.15: Read operation for UnQL [27]

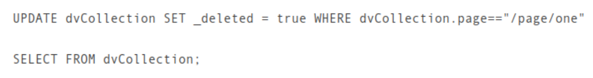
- **Update:**



```
UPDATE dvCollection SET dvCollection.author = "PEW", dvCollection.notes =
{this:"is", a:"test"}; WHERE dvCollection.page=="/page/one"
```

FIGURE 1.16: Update operation for UnQL [27]

- **Delete:**



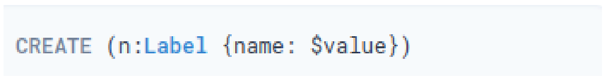
```
UPDATE dvCollection SET _deleted = true WHERE dvCollection.page=="/page/one"

SELECT FROM dvCollection;
```

FIGURE 1.17: Delete operation for UnQL [27]

1.5.4 Cypher

Cypher is a query language specifically designed for graph databases. It was first used with the Neo4j database management system. Cypher allows for simple and efficient implementation of creating and retrieving values from nodes in a graph database.
It is similar to SQL for graphs and was influenced by SQL, allowing you to concentrate on what data you want from the graph (not how to go get it) [29].

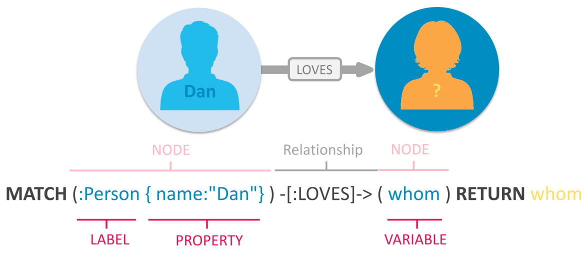
- **Create:**



```
CREATE (n:Label {name: $value})
```

FIGURE 1.18: Create operation for Cypher [30]

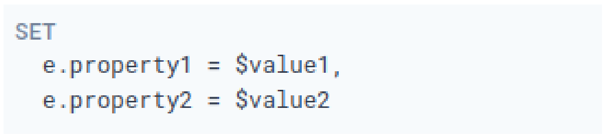
- **Read:**



```
MATCH (:Person { name:"Dan"} ) -[:LOVES]-> ( whom ) RETURN whom
```

FIGURE 1.19: Read operation for Cypher [30]

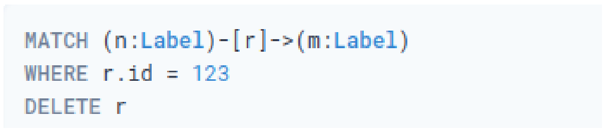
- **Update:**



```
SET
  e.property1 = $value1,
  e.property2 = $value2
```

FIGURE 1.20: Update operation for Cypher [30]

- **Delete:**



```
MATCH (n:Label)-[r]->(m:Label)
WHERE r.id = 123
DELETE r
```

FIGURE 1.21: Delete operation for Cypher [30]

1.6 Challenges of NoSQL Querying

The fact that NoSQL databases do not allow ACID (atomicity, consistency, isolation, and durability) transactions across multiple documents is one of the most commonly stated disadvantages of these databases. Single-record atomicity is suitable for many applications with proper schema design. Nevertheless, a lot of applications still need for ACID across several records[31].

Each brand of NoSQL database uses its own unique schema. In some cases (such as MongoDB), there is a flexible schema. In other situations (such as Redis), it is a pairs of key and value. In still other designs, it resembles relational databases (for example, Cassandra). The problem here is that each unique system has its own strengths and weaknesses, which must be learned before choosing the right NoSQL database for the situation at hand [32].

The absence of a standard query language in NoSQL databases presents a difficulty for developers who work with multiple databases. Unlike SQL databases that have a unified language (SQL), each NoSQL database has its own unique query language with different syntax, operators, and functions. This makes it hard for developers to switch between languages and increases development time and costs. Additionally, it also creates an obstacle for new users who must learn a new query language for each

NoSQL database they work with. Which in return rise the need for a standard query language that can be used and understood by everyone.

1.7 Conclusion

NoSQL databases have been a revolutionary invention that helped companies, scientists and programmers in general handle the huge volume of big data that has been gathered over the past years and it became the optimal choice for systems that don't have a fixed schema for their data.
One of those systems is the Question answering system (QA system) that's made for improving human-machine interactions by mapping natural language query to respective NoSQL data stores. So, QA systems for NoSQL datastores will be the topic of our next chapter.

CHAPTER 2 AN OVERVIEW OF QA SYSTEMS FOR NOSQL DATABASES

2.1 Introduction

The fast growth in the storage of huge volumes of information and the widespread usage of the internet allowed researchers to store and share data with the general public [2]. While most modern information retrieval systems do not directly answer our questions, they can be helpful in finding relevant documents that might contain the desired information. Instead of giving a concise answer, they provide the user with a list of papers to browse through in order to locate the necessary information. One skill that is thought to have a lot of promise is the ability to give clear answers to questions [33]. The Question-Answering system (QA system) allows users to access knowledge resources naturally by asking questions and receiving relevant and concise responses [2].

Question Answering (QA) is a research domain that combines various disciplines such as Information Retrieval (IR), Information Extraction (IE), and Natural Language Processing (NLP).[34]. Thus, with the diversity of NoSQL query languages, the development of a QA system for querying the NoSQL database is very suitable.

The aim of this chapter is to give an overview of QA systems with a bibliography study on deep learning of QA systems such as Bert, GPT, etc. the use of QA for NoSQL querying, especially, document-oriented databases of MongoDB.

2.2 Background

The Natural Language Interface to Databases (NLIDB) was one of the first question-answering systems that allowed users to input queries in natural language and retrieve responses from databases, reducing the need for formal query languages. BASEBALL and LUNAR were other notable Question-Answering (QA) systems that provided information on baseball leagues and soil samples, respectively, but were limited by their repository's insufficient domain information despite performing well using simple pattern matching techniques [35].

As research progressed in the field, QA systems shifted from simple pattern matching to linguistic analysis to understand the intended question requirements. MASQUE is an example of such a system that maps queries into a logic form and then into a database query for information retrieval. Semantic and statistical methods were also used for mapping by some researchers. There has been a shift toward open-domain QA systems in recent times[35].

In 1999, The TREC (Text REtreival Conference) is a popular information retrieval benchmark, which focuses on a list of different information retrieval (IR) research areas [36]. It began Open Domain QA, which has become an annual event to test the ability of QA systems to respond to different types of queries. These systems have developed into increasingly complicated systems that now use the internet as a data source. Factoid queries are the most common type of query that QA systems respond to, and there are several ways to handle them. Factoid QA systems output text, XML, or Wiki documents. The abundance of information readily available online is used by many QA systems to grow their knowledge bases and solve issues [35].

The QA systems provide a wealth of helpful information to internet consumers. Famous search engines such as Google and Bing frequently return documents linked to the terms entered by the user. These engines are open-domain question-answering systems.

Various approaches for developing QA systems start from the basic NLP (Natural Language Processing) algorithms-based approach towards the recently proposed methods of Deep Learning. NLP of question-answering systems frequently adopts an extractive strategy, with "a retriever" for retrieving relevant documents for the user's question, and "a reader" for extracting an answer from the documents it receives [37]. Before presenting the different approaches of deep learning for natural language processing of QA systems, we present in the next section the classification of QA systems according to various criteria.

2.3 Classification of Question-Answering systems

The Question answering systems can vary from one another depending on multiple factors, such as a domain, question types, data type, and approach [38].

2.3.1 QA systems based on application domain

For this type of classification, we notice two distinct types [39]:

- **Closed-domain:**
  Question-answering systems are designed to answer specific questions within a particular domain, such as sports, medicine, education, or entertainment. This task is relatively simple because NLP systems can rely on domain-specific knowledge that is often organized in ontologies. Closed-domain QASs are limited to a specific domain and cannot provide answers to questions outside that domain. Closed-domain can also refer to a situation where only certain types of

questions, such as those seeking descriptive information rather than procedural information, are considered.

- **Open-domain:**
  Question-answering systems can handle a wide range of questions from various domains, unlike closed-domain QAS which is limited to a specific domain. General QAS allows users to ask questions from any field, such as sports, politics, or education. These systems utilize global knowledge and ontologies, and they typically have access to a vast amount of data from which they retrieve the most appropriate answer.

2.3.2 QA systems based on the types of questions

The way questions are classified in question-answering systems has a direct impact on the accuracy of the generated answers, as different types of questions require different approaches. In 2003, the researchers found that 36.4% of errors in QAs are due to incorrect classification of questions. Other works use a content-based categorization method, but it is limited in its application to real-world questions. Another solution is based on a function-oriented approach that combines pattern matching and machine learning [38]. This classification system for QAs includes five categories: factoid, list, hypothetical, confirmation, and causal questions [38]. Each category is explained in detail in the following subsections.

2.3.2.1 Factoid type questions [what, when, which, who, how]

Factoid questions are simple and require a short phrase or sentence answer, often starting with a "wh" word, such as "who" or "what" [40]. QASs (Question Answering Systems) have shown satisfactory performance in answering factoid questions, with expected answer types generally being named entities that can be traced in documents through named entity tagging software [41]. This makes accuracy achievable, and Wikipedia or news wire text can be used as a source of information for QASs.

However, there are also cons to factoid questions, including the research issue of identifying and classifying them automatically. Additionally, other types of questions can be more challenging to answer. Descriptive questions ask for a definition or description of a term, fuzzy questions are imprecise, and relationship or information extraction (IE) questions identify the relationships between named entities. Dialogue questions can be incomplete or syntactically incorrect, and badly worded or ambiguous questions are challenging to process for generating accurate answers [42].

2.3.2.2 List questions

List questions in QASs require a series of entities or facts in their answers, such as "list the names of employees earning more than 5k." QASs treat these questions as a series of factoid questions that are asked one after the other, with previous answers being disregarded for subsequent questions. However, QASs often face difficulty in setting the threshold for the number or quantity of entities requested in list-type questions.

The benefits of list-type questions in QASs include the use of named entities as expected answer types, allowing for higher accuracy. Additionally, the techniques that work well for factoid-type questions can be applied to list-type questions. QASs also do not require extensive natural language processing to extract answers to list-type questions.

The downside of list-type questions in QASs is that determining the threshold for the number or quantity of entities requested can be problematic [40].

2.3.2.3 Hypothetical type questions

To ask about hypothetical events, one might pose a question that begins with "what would happen if." Hypothetical questions are subjective and cannot be definitively answered. QASs must use knowledge retrieval techniques to generate responses to these queries. The advantages of QASs answering hypothetical questions include catering to expert users who seek optimal solutions through common-sense reasoning and general knowledge. However, the downsides of hypothetical questions in QASs are that the expected answer type is uncertain, making the accuracy of QASs low. Factoid QAS techniques do not translate well to hypothetical questions, and reliability is contingent upon the user and context [41].

2.3.2.4 Causal questions [how or why]

Causal questions seek explanations about an entity or event, and the answers are typically not named entities, as in factoid-type questions. QASs require advanced natural language processing techniques to analyze the text at the pragmatic and discourse level to generate answers [43].

The advantages of causal-type questions in QASs include providing users with explanations, reasons, and elaborations related to specific events or objects. The disadvantages of causal-type questions are the problems in determining relevant or unique answers, and problems related to efficient retrieval models, which can cause problems due to polysemy, homonymy, and synonymy. Furthermore, "why" questions often have subjective answers, which can range from sentences to paragraphs. The identification of discourse relationships in source documents is required to generate answers to such questions [44] [45] [46].

2.3.2.5 Confirmation questions

Confirmation questions in QASs require answers in a binary form, either "yes" or "no". To generate such answers, the systems require advanced inference mechanisms, world knowledge, and common sense reasoning.

The advantages of confirmation-type questions in QASs include the potential for expert users to search for new knowledge that requires these advanced reasoning techniques.

However, the downside is that answering confirmation questions accurately requires a higher level of knowledge acquisition and retrieval techniques, which are still under development [38] .

2.3.3 QA systems based on data type

We classify QASs based on the data type present in the source text. The different categories are (1) structured data sources, (2) unstructured, and (3) semi-structured data sources.

2.3.3.1 Structured data source

Structured documents arrange data into semantic sets, with similar entities grouped into relations, where each entity within a relation shares the same attributes. A schema describes all the entities within a unit, and the data is organized in a defined format. When a query is matched with structured data, it is an exact match, and a query language is used.

The advantages of using structured data sources in QASs include higher reliability of answers, as the correct information is stored in the data source, and QASs do not require complex natural language processing of these data sources. However, based on the literature reviewed, the drawbacks of structured data sources in QASs are that they may have limited information stored, reference reconciliation may be necessary, building the data source can be labor-intensive, and different structured data sources have varying representations and accept other query languages, which requires the system to transform questions into queries based on the type of data source being used [38].

2.3.3.2 Unstructured Data source

Data in an unstructured data source can be of any type and is not arranged in a semantic set with no strict rules governing how it should be organized. QAS which handles unstructured data requires the use of natural language processing and information retrieval technologies to find answers.

The advantage of using unstructured data sources in QASs is that information can be easily added or updated. However, the drawbacks of unstructured data sources in

QASs are that the representation of unstructured data sources can be problematic, the reliability of answers is low, and paraphrasing is more prevalent[38].

2.3.3.3 Semi-structured data source

In a semi-structured data source, there is no clear separation between the stored data and the schema. The advantages of using semi-structured data sources in QASs include the representation of information in data sources constrained by a schema, providing a flexible format for exchanging data between different databases, and the ability to transform structured data into semi-structured formats for web browsing purposes. However, the drawbacks of semi-structured data sources in QASs are that reference reconciliation may be necessary, and building the data source can be labor-intensive [38].

2.3.4 QA systems based on approach

2.3.4.1 Pattern matching approach

- **The pattern-matching approach** is a technique that focuses on the communicative effect of text patterns, and it differs from other computing methods that use complex processing. Instead, many QA systems learn text structures from passages automatically, making them simpler and more favorable for small implementation and use. In pattern-matching QA systems, surface text patterns are often used to find answers to factual questions, while templates are used for closed-domain systems. Some systems rely solely on surface patterns, while others use templates as a response generator [35].

- **The surface Pattern approach** matches responses from the surface structure of retrieved texts using a comprehensive list of patterns, which can be manually crafted or generated automatically. The solution to a query is determined based on the similarity between the patterns and their associated semantics. The approach offers high precision with the level of human involvement in creating the set of patterns [35].

- **The Template-based approach** uses pre-formatted patterns for questions and focuses on illustration rather than interpretation of questions and answers. The set of templates is designed to cover the problem domain with an optimal number of templates, with each template representing a range of questions of its type. Entity slots in the templates include missing elements that must be filled to generate the query template and retrieve the corresponding response from the database. The response returned by the query is raw data, which is sent to the user. This approach is similar to an automated Frequently Asked Questions (FAQ) system that provides pre-stored answers to user questions. However, unlike static FAQs, question templates are filled dynamically with parameters [33].

2.3.4.2 Linguistic based approach

To create a question-answering system, it is necessary to have an understanding of natural language text, linguistics, and general knowledge. In the past, researchers have relied on artificial intelligence methods that combine natural language processing techniques and a knowledge base or corpus to construct QA logic. The information in the knowledge base is organized in various forms such as production rules, logics, frames, templates (represented as triple relations), ontologies, and semantic networks. These are used to analyze the question-answer pair.

Linguistic techniques like tokenization, POS tagging, and parsing are utilized to formulate the user's question into a precise query that extracts the relevant response from a structured database. Building an appropriate knowledge base is a time-consuming process, so these systems are usually used for problems that have long-term information needs for a specific domain [33].

2.3.4.3 Statistical Approach

The availability of vast amounts of data on the internet has increased the significance of statistical methods. These methods are particularly useful because they can handle very large amounts of data and the diversities (such as heterogeneity) that the data may exhibit. Unlike structured query languages, statistical methods can generate questions in natural language form. However, this method requires a substantial volume of data to achieve precise statistical learning [35].

Generally, statistical approaches have been successfully utilized in multiple stages of a Question Answering (QA) system. For instance, methods like Support Vector Machine (SVM) classifiers, Bayesian classifiers, and Maximum Entropy models have been explicitly applied to classify questions.

These statistical approaches evaluate questions to anticipate the type of answer the user is seeking. The models are trained using a corpus of questions or documents that have been labeled with specific categories within the system [33].

2.4 Deep learning for the NLP of QA

There are different approaches to deep learning for the natural language processing of QA systems. In this section, we will talk about the known Deep Learning Neural Networks(DLNN) with a focus on the revolutionary ones, such as Transformer architecture and the pre-trained models base on it (Bert, T5, GPT...) since they are the most dominant when it comes to NLP tasks.

2.4.1 Transformer model

In the paper "Attention is All You Need" [47], the authors propose a new deep learning method of processing data sequences through the "Transformer model".

The Transformer model uses self-attention mechanisms instead of traditional recurrent or convolutional layers. This self-attention mechanism allows the model to learn the relationships between elements in the input sequence (natural language), making it a key innovation in the field of deep learning for processing sequences of data with more parallelization, which led to the development of pre-trained systems such as BERT with its variants, T5, GPT, which are trained with large language datasets, such as the Wikipedia Corpus, and can be fine-tuned for specific tasks.

A set of hidden states that describe the dependencies between the input sequence's elements are produced by the Transformer after the input sequence is transformed into a high-dimensional space and processed through several self-attention layers. Predictions about the objective job, such as machine translation or text categorization, are then made using these hidden states. The result of the authors' work [47] showed that the Transformer surpasses the performance of prior state-of-the-art models that utilized recurrent and convolutional neural networks. Additionally, the authors highlight the efficiency and parallelizability of the Transformer, making it easier to train compared to previous models.

The work carried out by **Vaswani et al** [47] is regarded as a turning point in deep learning and has had a significant impact on the creation of NLP models. Its fundamental concepts have been extensively used and developed in a number of later works. The main architecture of this model is displayed in Fig 2.1.

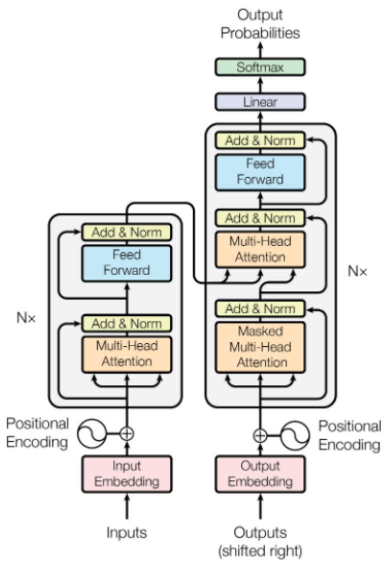


FIGURE 2.1: The Transformer architecture [47]

2.4.2 BERT pre-trained model

Jacob Devlin et al. [48] introduce a new pre-training approach for natural language processing (NLP) called BERT (Bidirectional Encoder Representations from Transformers). The idea is to train a deep bidirectional representation model on a large corpus of unlabelled text (ex. Wikipedia corpus), and then fine-tune it for specific NLP tasks. Let's talk about the main points of BERT:

- **Architecture:** BERT is a transformer-based neural network architecture, similar to the one introduced by Vaswani et al. [48]. However, BERT is designed to be bidirectional, meaning it can take into account the entire context of a word in a sentence, rather than just the words that come before it or after it. BERT uses a multi-layer bidirectional transformer encoder to generate a sequence of contextualized word embeddings. It also introduces two novel training tasks: masked language modeling and next sentence prediction, which help the model learn more about the relationships between words in a sentence and between sentences (see Fig 2.2).

- **Implementation:** The authors trained BERT on a large corpus of unlabelled text, consisting of 3.3 billion words from various sources, including books and web pages. A pre-training objective was used that involves randomly masking some of the words in the input sequence and asking the model to predict the masked words. The next sentence prediction task was later introduced, which involves feeding two sentences to the model and asking it to predict whether the second sentence is the next sentence in the original text. This task helps the model learn more about the relationships between sentences. BERT was fine-tuned on a range of target NLP tasks, including question answering, sentiment analysis, and named entity recognition, achieving state-of-the-art results on many of them.

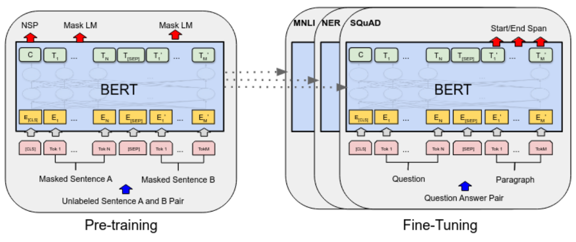


FIGURE 2.2: Overall pre-training and fine-tuning procedures for BERT. [48]

- **Results:** The authors showed that BERT consistently outperforms them on a range of target NLP tasks. For example, on the GLUE (General Language Understanding Evaluation) benchmark, which measures performance on a range of NLP tasks, BERT achieved state-of-the-art results on all 9 tasks. On the SQuAD benchmark, which measures question-answering performance, BERT achieved a new state-of-the-art score.

The work concludes that BERT is a powerful pre-training method for NLP tasks thanks to its bidirectional architecture and innovative pre-training tasks. Furthermore, it is mentioned that BERT is a flexible and robust tool for NLP researchers as it can be fine-tuned on various downstream tasks with minimal modifications to the task-specific architecture.

2.4.3 RoBERTa pre-trained model

In 2019, **Yinhan Liu et al.** [49], researchers at Facebook AI, present an optimized version of BERT, which is a popular pre-training approach for natural language processing (NLP) tasks. The principal goal of this new version called RoBERTa for Robustly Optimized BERT Approach is to improve the performance of BERT on a wide range of NLP tasks and make it more robust to domain shifts and different task types.

The main contributions of these researchers are the adjustments made to the original BERT training procedure to create RoBERTa:

First, the next sentence prediction (NSP) task got removed because it was found to be ineffective in the original BERT model. Instead, RoBERTa was trained on longer sequences of text, up to 512 tokens, which resulted in a better contextual understanding of the text.

Second, RoBERTa uses dynamic masking during training, which means that different tokens are masked each time a sequence is presented to the model during training. This is in contrast to the static masking used in BERT, where the same tokens are always masked during training. Dynamic masking improves the model's ability to handle out-of-vocabulary (OOV) words and unseen data.

Finally, RoBERTa was trained on a larger volume of data than BERT (a dataset of 160GB of text, which is more than 10 times larger than the dataset used to train BERT). The increase in the amount of training data improved the model's robustness and generalization ability even further.

2.4.4 DistilBERT pre-trained model

Sanh et al. [50] proposed a new model called DistilBERT, which is a distilled version of BERT. DistilBERT is smaller, faster, and cheaper than BERT, while still maintaining similar performance on a variety of natural language processing tasks.

The authors use a technique called distillation, which involves training a smaller model to mimic the behavior of a larger, more complex model. The resulting DistilBERT model is much smaller than BERT, with 40% fewer parameters, a 60% reduction in memory usage, and 97% of the language understanding capabilities retained. DistilBERT has neither token_type_ids to indicate which token belongs to which segment, nor options to select the input positions (position_id input).

Despite its smaller size, DistilBERT achieved similar performance to BERT on several benchmark tasks, including question answering and text classification. Besides its smaller size, DistilBERT is also faster than BERT, with inference times that are two to five times faster, depending on the task.

2.4.5 DeBERTa pre-trained model

In 2020, **He et al.** [51] propose a new transformer-based language model called DeBERTa (Decoding-enhanced BERT with Disentangled Attention) to improve upon the popular BERT model by incorporating a number of new features and techniques (Disentangled attention and Enhanced mask decoder). In contrast to BERT, which uses a vector to represent each word in the input layer by combining its content and position embedding, DeBERTa utilizes two vectors to represent each word separately, one for its content and another for its position. The attention weights among the words are then calculated based on disentangled matrices that consider their contents and relative positions. The reason for this approach is the realization that the attention weight between a pair of words depends not only on their contents but also on their relative positions. For example, the relationship between "deep" and "learning" is stronger when they are adjacent to each other compared to when they are present in different sentences.

DeBERTa, like BERT, is pre-trained using masked language modeling (MLM) which is a fill-in-the-blank task, where a model is taught to use the words surrounding a mask token to predict what the masked word should be. However, DeBERTa enhances the mask decoder by incorporating absolute word position embeddings right before the softmax layer where the model decodes the masked words based on the aggregated contextual embeddings of word contents and positions.

The absolute position of words in a sentence is often crucial for prediction because words' syntactic roles depend on their absolute positions in the sentence. For example, consider the sentence "a new store opened beside the new mall" with the words "store" and "mall" masked for prediction. Although the local contexts of the two words are similar, they play different syntactic roles in the sentence. Therefore, DeBERTa takes into account a word's absolute position in the language modeling process by incorporating absolute word position embeddings in the mask decoder.

This enhancement improves the efficiency of pre-training and the performance of downstream tasks. In the NLP tasks, compared to RoBERTa-Large, a DeBERTa model trained on half the training data performs consistently better on a wide range of NLP tasks, achieving improvements on MNLI (Multi-Genre Natural Language Inference), SQuAD v2.0, and RACE (ReAding ComprEhension) benchmarks.

Overall, DeBERTa outperforms BERT on a variety of benchmark tasks, including question answering, natural language inference, and text classification. The authors believe that the use of disentangled attention and other new techniques could be useful for improving the performance of other transformer-based models as well.

2.4.6 T5 pre-trained model

In 2019,**Colin Raffel et al** published the paper [52] which presents a new approach to transformer architecture called the "Text-to-Text Transformer" (T5). The authors propose that casting all NLP tasks as "text-to-text" problems allows for T5 to be trained on a large amount of text data in a "multi-task" setting, where it learns to generate a diverse set of output sequences for different input sequences.

- **Methodology:** The authors train T5 on a large corpus of text data and evaluate its performance on a variety of NLP tasks, including machine translation, question answering, and summarization. They also explore the limits of transformer architecture with T5 by examining how the performance of the model varies with the amount of training data and the size of the model. Additionally, the authors investigate the effectiveness of fine-tuning T5 on smaller datasets for specific tasks.

- **Evaluation:** The experimental results demonstrate that T5 outperforms previous state-of-the-art models on a variety of NLP benchmarks, including SuperGLUE, a challenging benchmark dataset for evaluating the performance of natural language understanding models. The authors also find that T5 exhibits strong performance even with small amounts of training data and that increasing the size of the model can further improve performance, but with diminishing returns. The results also show that fine-tuning T5 on smaller datasets for specific tasks can achieve competitive performance.

T5 is a novel approach for NLP tasks that achieves state-of-the-art performance on a variety of benchmarks. It offers a unified approach to the transformer model that can be applied across a wide range of tasks. The authors suggest that further exploration is needed to determine the best fine-tuning strategies for specific tasks, but overall, T5 represents a significant step forward in the field of NLP.

2.4.7 OpenAI GPT-2 pre-trained model

In 2019, **Radford, Alec, et al.** from OpenAI[53] introduced the concept of pre-training large-scale language models using self-supervised learning objectives as a way to learn general-purpose representations of natural language. The authors. They argue that this approach can address the problem of data scarcity in many NLP tasks, where annotated training data is limited, and can lead to significant improvements in model performance.

- **Methodology:** The authors present GPT-2 (Generative Pre-trained Transformer), a transformer-based language model that has been pre-trained on a diverse corpus of text data using a self-supervised learning objective. The model architecture consists of a multi-layer transformer encoder that is trained to predict the next word in a sentence given the previous words.

  The training approach of GPT-2 involves training it on a large corpus of text data using a masked language modeling objective, where some words in the input are randomly masked out and the model is trained to predict the masked words based on the context. They also introduce a new training objective, called "unsupervised paraphrasing", which involves training the model to generate alternative phrasings of a given sentence.

- **Evaluation:** The authors evaluate the performance of the GPT-2 model on several NLP tasks, such as language modeling, text generation, summarization, and machine translation. They show that the model achieves state-of-the-art performance on these tasks, without requiring task-specific training data or supervision. They introduce a new benchmark dataset, called LAMBADA, for evaluating the ability of language models to perform language understanding tasks. The dataset consists of passages of text, where the task is to predict the last word in the passage based on the context. They show that the GPT-2 model outperforms several baseline models on the LAMBADA dataset, demonstrating its ability to capture long-term dependencies and semantic relationships in text.

GPT models can lead to significant improvements in performance on diverse NLP tasks, including language translation, QA, and text completion. They can enable new applications in natural language understanding and generation.

GPT-3 is the advanced GPT model, which has 175 billion parameters, whereas GPT-2 has only 1.5 billion parameters.

GPT-4 is a large multimodal model that can accept both image and text inputs and generate text outputs. It can generate, edit, and iterate with users on creative and

technical writing tasks, such as composing songs, writing screenplays, or learning a user's writing style.
OpenAI may release new versions of the GPT model in the future with more and more parameters and sources.

2.4.8 Bart pre-trained model

Lewis et al. [54] introduce a new pre-training method for natural language processing called BART, which stands for "Bidirectional and Auto-Regressive Transformer". BART is a sequence-to-sequence model that is trained to reconstruct corrupted text by denoising it.
The BART model is a modified version of the Transformer architecture used in other sequence-to-sequence models like GPT and BERT. BART is bidirectional, meaning it can generate output both from left-to-right and right-to-left, and it is also auto-regressive, meaning that it generates output one token at a time conditioned on the previous tokens it has generated.

To pre-train BART, the authors use a denoising auto-encoder approach. They corrupt the input text by randomly masking some of the tokens, adding random noise, and shuffling the remaining tokens. The model is then trained to reconstruct the original sequence from the corrupted version. This pre-training approach is similar to the masked language modeling objective used in BERT, but with additional noise injection. The authors show that this pre-training method can be used to improve the performance of various natural language tasks, including text generation, translation, and comprehension.

One notable contribution is the introduction of the XSUM dataset, which consists of short news articles and their corresponding one-sentence summaries. The authors demonstrate that BART outperforms previous state-of-the-art models on this dataset, achieving a new state-of-the-art performance on both the summarization and sentence compression tasks.

Overall, the work presents a promising new pre-training method for natural language processing that can be applied to a wide range of tasks and introduces a new benchmark dataset for summarization and sentence compression.

2.5 Related works on QA of databases

Question-answering systems have revolutionized the way individuals retrieve information by enabling them to obtain answers to their inquiries by asking in everyday language. As the volume of data being produced continues to grow, NoSQL databases have become increasingly crucial as they offer the ability to manage big data and adjust to changing demands. Despite the prevalence of NoSQL databases, there has been limited advancement in creating question-answering systems that are compatible with this type of database. One of the key challenges in developing question-answering systems for NoSQL databases is the diversity of query languages used in these databases. Unlike SQL databases that use a standard query language, NoSQL databases often employ their own distinct query languages, making it hard for question-answering systems to provide a uniform experience while supporting multiple databases.
In this section, we classify major efforts in the area of QA systems for databases into two classes: the ones that use SQL databases and works that use NoSQL databases.

2.5.1 QA for SQL databases

Several research works have been proposed for QA of SQL databases, which lead to translating questions in natural language to SQL queries such as dependency parsing, Recurrent Neural Networks of deep learning, Seq2Seq, Attention-mechanism, etc.

Ahkouk, Karam, et al. [55] proposed a new approach to generate SQL queries from sketches. They note that SQL queries are often complex and difficult for non-experts to write. To address this issue, they propose a method that allows users to draw sketches of the desired query, rather than requiring them to write the query directly.

The SQLSketch approach consists of three main steps:

1. Sketch parsing: The user's sketch is parsed to identify its components, such as tables, attributes, and conditions.

2. Query generation: The components identified in step 1 are mapped to corresponding SQL expressions to generate a candidate query.

3. Query ranking: The candidate query is ranked based on its similarity to the user's intent, and the top-ranked query is returned as the final result.

The authors evaluate their approach on a dataset of sketches and compare it to existing sketch-based query generation methods. They demonstrate that SQLSketch outperforms these methods in terms of accuracy and efficiency. Overall, the work provides an interesting and innovative approach to generate SQL queries that could potentially make the process more accessible to non-experts.

Rami Reddy et al [56] presented a dialogue-based question-answering (QA) system for the Telugu language. The system combines different machine learning methods, including natural language processing, deep learning, and semantic similarity measures together with the purpose of delivering precise and relevant responses to user questions. The system is designed to comprehend the user's queries which are expressed in natural language and extract appropriate responses from a knowledge base. The authors developed a corpus in the Telugu language that includes numerous questions and their corresponding answers. This corpus was used to train and evaluate the QA system. The proposed system's performance was judged by using different evaluation metrics which include precision, recall, and F1 score. The result shows that the system functions effectively in terms of the relevance and precision of the responses provided to the user. The authors also address the obstacles encountered while developing a Telugu-based QA system and propose future research directions to improve the performance of the system. In summary, the approach outlines an efficient method to create a conversation-based QA system in Telugu that can be extended to other languages.

Ping Wang et al [57] focused on addressing the difficulty in retrieving patient information from electronic medical records (EMR) by using question-to-SQL generation methods. However, most existing approaches are not suitable for the healthcare domain because of the lack of a dedicated healthcare Question-to-SQL dataset and the complexity introduced by abbreviations and typos in questions. The authors propose a deep learning-based model called TREQS that directly generates the SQL query for a given question and performs necessary edits using an attentive-copying mechanism and task-specific look-up tables. The authors also create a new large-scale Question-SQL pair dataset called MIMICSQL specifically for the healthcare domain.

Victor Zhong et al. [58] present Seq2SQL, a sequence-to-sequence neural network that translates natural language questions to their corresponding SQL queries. This model is based on an attention mechanism to generate a sequence of SQL tokens that correspond to the input question. The model is trained using a combination of supervised and reinforcement learning, where the latter is used to optimize the model for the final query execution accuracy.

The proposed approach includes an encoder-decoder model and a reinforcement learning (RL) model based on policy (see Fig 2.3). The encoder-decoder model takes as input a natural language question and generates a sequence of tokens that represent a SQL query. The RL model interacts with a database, executes the query, and receives a reward signal based on the precision of the outcome. By optimizing the expected reward, the RL model fine-tunes the encoder-decoder model's parameters.

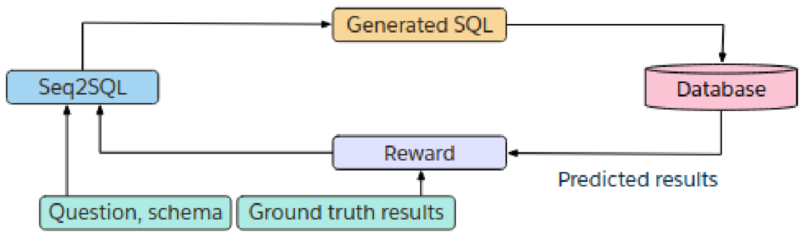


FIGURE 2.3: The work of Seq2SQl-based model [58]

The proposed method achieves state-of-the-art results on the WikiSQL benchmark dataset, which contains over 24,000 examples of natural language questions and their corresponding SQL queries. The authors have examined the impact of various model components, such as attention mechanisms and the size of the reinforcement learning training set, on performance. Additionally, they have conducted experiments to analyze the ability of the model to handle different types of questions and databases.

In 2017, **Xiaojun Xu et al** [59] present a neural network-based approach for question answering over relational databases without the need of reinforcement learning. The proposed model consists of two components:

1. *The question encoder:* uses a bidirectional LSTM network to encode the input question into a vector representation.

2. *The SQL generator:* is a sequence-to-sequence model that takes the question representation and generates a sequence of tokens corresponding to the SQL query.

The model also incorporates a set of SQL grammar rules to ensure that the generated queries are valid. The authors evaluate the performance of their model on the Spider dataset and show that it outperforms existing approaches in terms of accuracy and efficiency.

2.5.2 QA for NoSQL databases

In this section, we are going to discuss relative works for QA systems for NoSQL databases. To the best of our knowledge, there are few efforts regarding the issue for QA systems of NoSQL databases.

Sebastian Blank et al [60] proposed a trainable question-answering system that enables users to access information stored in a NoSQL database using natural language. The system overcomes a major obstacle, the non-differentiability of database operations, by using policy-based reinforcement learning. The performance of the system was evaluated using Facebook's bAbI Movie Dialog dataset and it performed well, with a score of 84.2% compared to other benchmark models. The authors believe that the system is ideal for practical situations where knowledge is stored in external databases and acquiring intermediate labels is too expensive for other non-end-to-end trainable question-answering systems.

The system is based on the proposed SeqPolicyNet model, which is designed to solve the KBQueryBot task for converting a natural language query into the domain-specific query language of an external knowledge source (KB). The main component of the model is an attention-based pointer network, which fills the slots in a predefined Elasticsearch query template. The output sequence is composed of elements that either select a column name of the knowledge base (KB) or point to a token of the question for entity extraction. These outputs are referred to as "selection" or "extraction outputs." The extraction outputs serve as start and end pointers, allowing the model to extract an entity composed of a span of tokens in the question. For example, given the question "Who was the director of Gone With the Wind?" the start pointer would be "Gone" and the end pointer would be "Wind." The model can be seen as a pointer network for entity extraction with reduced complexity.

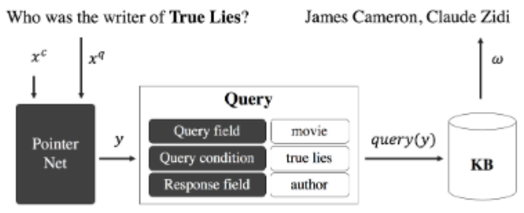


FIGURE 2.4: System architecture of SeqPolicyNet [60]

This work used Elasticsearch (ES) as a NoSQL document-oriented database that drives the search functions of many modern websites. ES uses a JSON-based query language to formulate a query template. To evaluate the effectiveness of this proposal, the authors used two metrics: the accuracy of valid queries (Accvq), which represents the percentage of questions that generated a valid query, and the accuracy of correct results (Acccr), which refers to the percentage of questions where they retrieved the correct answer.

Another interesting work is of **Do et al.** [61], who developed a QA system for the tourism industry in Vietnam that blends knowledge graphs and deep learning techniques. The system starts by addressing a user's inquiry about a particular place in Vietnam and then goes on to provide comprehensive information, including when the place was discovered and the reason behind its name. The QA system also offers recommendations for related tourist destinations. The system employs deep learning to produce simple natural language answers, while the knowledge graph is used to identify relevant entities and generate a list of natural language answers. The research was based on a manually created dataset gathered from Vietnamese tourism websites. The results demonstrate that the QA system combining the two approaches provides more information than previous systems and achieved an F1 score of 0.83 and a precision of 0.87 on the test set.

The system has seven components, as shown in Fig. 2.5. The first step is for the user to input a question about visiting places in Vietnam. The system then transforms the question into one-hot encoded vectors, which serve as inputs for the LSTM model. The LSTM model searches its knowledge base for the question and outputs two branches: a simple natural language answer and a graph query. The simple answer is a question-answer pair that has been trained by the LSTM model. The graph query accesses the Neo4j knowledge graph database to produce a list of natural language answers.

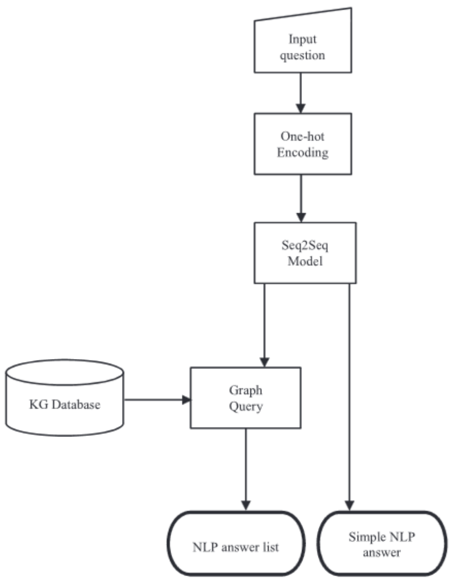


FIGURE 2.5: The flow chart of the Vietnamese tourism question answering system [61].

Pradeep et al [62] proposed the conversion of English queries to MongoDB queries using an Encoder-Decoder-based machine translation method. The latter involves creating a thought vector of the given English query using an Encoder neural network and predicting the MongoDB query using a Decoder neural network. The proposed system uses ten separate deep learning models to handle ten types of MongoDB queries and shows satisfactory results. The proposed system takes an English query as input and converts it into the appropriate MongoDB format (see Fig 2.6).

The system has a question type detection module that identifies the type of English query and routes it to the appropriate model. The selected model then generates the corresponding MongoDB query. The proposed system handles ten different types of MongoDB queries and has an average accuracy rate of 71.5%.

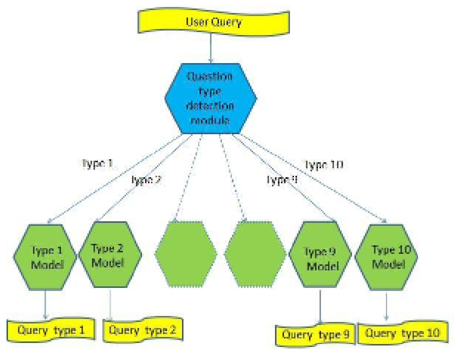


FIGURE 2.6: Architecture of conversion system of English queries to MongoDB queries [62].

Gadekar et al. [63] proposed a novel approach to querying and manipulating data in MongoDB using natural language commands. This approach consists of an interface that can understand and translate natural language queries into MongoDB commands, making it easier for non-technical users to interact with the database. The system is built using Python and the Natural Language Toolkit (NLTK), and it includes modules for natural language processing, query parsing, and database communication. The authors describe the architecture of the system and the algorithms used for query parsing and database communication.
Overall, the work presents an innovative approach to interacting with databases using natural language queries, which could potentially make it easier for non-technical users to retrieve and manipulate data.

2.6 Discussion

Most QA systems for databases are built for SQL language and based on machine-translation techniques on questions written in natural language to transform them into SQL queries for example, Sketch parsing, Seq2SQL, LSTM deep learning, etc. Unfortunately, very little research has investigated to study of the NoSQL language rather than SQL. The existing works are based on deep learning and NLP techniques. However, these approaches are based on the external source, which is the knowledge graph in order to identify relevant entities and generate a list of natural language answers. Besides, some works are limited to some queries and required more details about the architecture and the dataset used.

The use of NoSQL databases, especially the popular document-oriented database MongoDB for question-answering systems is still a relatively new area of research. Despite the increasing popularity of NoSQL databases for handling big data and semi-structured data, there has been limited research on their use for question-answering systems. This is due in part to the fact that traditional question-answering systems have largely relied on relational databases and structured data, rather than unstructured data commonly stored in NoSQL databases.

In terms of datasets for natural language and their corresponding representation in MongoDB, there is a lack of publicly available datasets specifically designed for this purpose. The lack of well-established and widely-used datasets for natural language in NoSQL databases can make it difficult for researchers to study and evaluate the use of MongoDB for question-answering systems. And for this reason, to advance our work on the QA system, we opted to create our own "Question to MongoDB Query" dataset for a specific domain.

The main objective of this Master's thesis is to propose a new Question-answering system to facilitate the querying of the MongoDB database to non-experts by using natural language.

2.7 Conclusion

Question-answering systems are designed to enable users to ask natural language questions and receive accurate answers. They involve processing large amounts of data, including unstructured data, and using various techniques such as natural language processing, information retrieval, machine learning, deep learning, etc. They can be used in various fields and provide an efficient and effective means of accessing information. Designing a high-performing question-answering system can be challenging, and ensuring accuracy and reliability is crucial. Despite the challenges, question-answering systems continue to evolve and improve due to advances in technology and the growing data volume modelled in the NoSQL approach with the need for fast and accurate access to information using natural language.

Finally, for our next chapter, we will propose a QA system for MongoDB Database. A Question Answering (QA) system for MongoDB enables users to retrieve answers to their queries through natural language access to the database. Deep Learning models are used to improve the performance of these systems by processing large amounts of unstructured data and learning complex patterns. Implementing a QA system for MongoDB using Deep Learning requires training the system on a diverse dataset of questions and answers (Keep in mind there is a lack of a publicly shared Dataset of this kind which required making one from scratch), integrating it with MongoDB, designing it to handle natural language queries, and evaluating its performance using metrics. A QA system for MongoDB using Deep Learning can enhance the accuracy and flexibility of a user's data access experience.

CHAPTER 3 DEEP LEARNING-BASED QA FOR MONGODB DATABASE

3.1 Introduction

In the previous chapter, we discussed the challenges of developing question-answering systems for NoSQL databases. Despite the increasing popularity of NoSQL databases, there is a noticeable lack of research work related to developing question-answering systems for these databases. We also annotate the lack of related datasets for applying the machine learning algorithms. Therefore, in this chapter, we aim to address this gap by proposing a novel deep learning-based approach for building a question-answering system for NoSQL databases.

Our proposed approach involves for the first time, the construction of a new large-scale dataset, **MongoQpedia** (Mongo Query-pedia) for a deep learning model. This task requires a lot of time and effort for obtaining a high-quality dataset, which will be used for training and testing our deep-learning model of the QA system.

Moreover, we introduce the conceptual architecture of our deep learning model, called "**NoT5QL**" (NoT5QL is a clever wordplay on the term 'NoSQL', incorporating 'T5' in the middle and replacing the 'S' in SQL with a '5', resulting in a phrase that can be read as "NotSQL".), which involves fine-tuning **T5**, a pre-trained transformer model. Our proposed approach is expected to achieve better performance compared to traditional approaches that rely on manually crafted rules or keyword matching.

Overall, this chapter provides a detailed contribution, including the MongoQpedia dataset creation process,augmentation techniques used (including algorithems), and the conceptual architecture of our NoT5QL model.

3.2 Motivation and Contributions

With the increasing popularity of NoSQL databases to store and manage large-scale data, it is becoming crucial to have a system that can help users easily retrieve information from these databases by using natural language. A Question-Answering (QA) systems allow users to query an external database by using natural language. Developing a QA system for NoSQL databases is a good solution, especially when implementing a deep learning approach to improve the performance of such a system. However, the lack of research works in this area has resulted in limited tools and techniques for building QA systems for NoSQL databases.

Our research work aims to fill this gap by proposing a novel approach for building a question-answering system for the NoSQL document-oriented database of MongoDB. One of the main reasons for selecting MongoDB is its popularity among developers and businesses worldwide.

Our approach involves creating a dataset for a deep learning model, using data augmentation techniques to increase the dataset size, and fine-tuning a pre-trained transformer model for the task of the dynamic Question Answering related to MongoDB databases.

The major contributions of this work are enlisted as follows:

- **MongoQpedia Dataset Creation:** Our proposed method involves generating a high-quality dataset called MongoQpedia for training a question-answering system. The dataset is meticulously created by extracting relevant information from an established MongoDB database and formulating a comprehensive set of questions along with corresponding answers. By leveraging this dataset, the resulting question-answering system will possess a deep understanding of the data and be better equipped to provide accurate responses to complex queries.

- **MongoQpedia Dataset validation:** By conducting a rigorous validation process, we were able to ensure that the data was of high quality, reducing the likelihood that future researchers would need to spend significant time cleaning or correcting the data. This would save time and resources, and allow researchers to focus on analyzing the data and drawing meaningful insights.

- **Data Augmentation Techniques:** The accuracy and effectiveness of the deep learning model are significantly influenced by the amount and quality of the dataset. Hence, we created a data augmentation pipeline that incorporates several data augmentation techniques aimed at expanding the dataset size while preserving a high level of quality. These techniques include paraphrasing, back-translation, and named entity replacement. By using paraphrasing, we generate additional instances of the original data by rewording the existing sentences while preserving their meaning. Back-translation is another technique we employ, which involves translating the data into another language and then translating it back into the original language to create new data samples. We also use named entity replacement to substitute named entities in the text with similar entities, which further increases the diversity of the dataset. Overall, by implementing these data augmentation techniques, we aim to improve the model's performance and make it more robust to varying inputs.

- **Fine-tuning of T5 Pretrained Transformer Model:** we present a novel conceptual architecture for fine-tuning the T5 pre-trained transformer model [52] to answer questions related to MongoDB databases. Fine-tuning is a crucial step in customizing the pre-trained model to our specific task, and involves training the model on our dataset to improve its performance on our particular problem domain. By fine-tuning T5, we aim to create a high-performing and efficient question-answering system that can accurately respond to complex queries related to MongoDB databases. Our proposed architecture provides a solid framework for fine-tuning the T5 model, and we expect that it will have a significant impact on the performance of question-answering systems in the field of database management.

- **Experimental Results:** We evaluated the effectiveness of our proposed approach by conducting experiments on a test split of the dataset. Our experiments demonstrate that our approach achieves high accuracy in answering questions related to the test split (more details in Chapter Four). By fine-tuning the T5 model on our specific dataset, we were able to train it to accurately capture the nuances of the test data, enabling it to provide precise and reliable responses to various queries. To evaluate the performance of our approach, we used several standard evaluation metrics. Our experimental results show that our approach has significant potential for efficient and effective question-answering systems, as it achieved notable accuracy. These findings suggest that our approach can have a significant impact on various applications that require the ability to accurately answer complex questions.

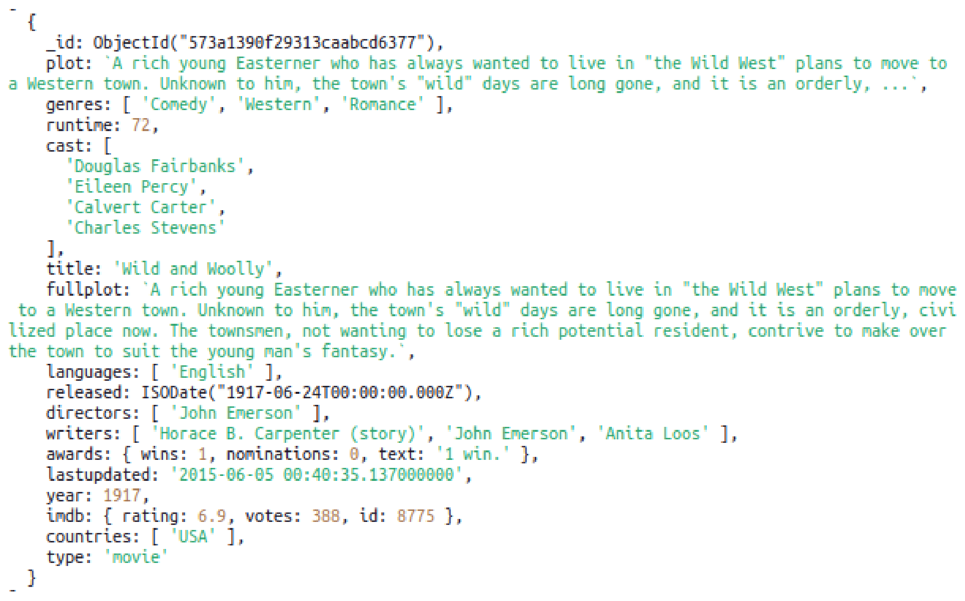
Overall, our proposed approach can serve as a useful tool for retrieving information from NoSQL databases, and our contributions can pave the way for future research in this area.

3.3 MongoQpedia dataset Construction

Our dataset-building strategy involves extracting information from a real-world NoSQL database, specifically the Mflix sample movies database provided by MongoDB. This database is extensive and complex, containing information about various entities such as movies, TV shows, actors, directors, and producers [64]. Our dataset primarily focuses on questions related to movies and actors. By utilizing the Mflix database specifically the movies collection, we were able to create a diverse and comprehensive dataset that accurately reflects the nuances of real-world data. Our dataset enables the fine-tuned T5 model to capture the complexities of the Mflix database and accurately answer complex queries related to movies and actors. The use of a real-world NoSQL database in our dataset construction strategy provides practical relevance to our approach and ensures that the model can be effectively applied to real-world problems.

3.3.1 Description of Mflix database

The Mflix movies database is a document-oriented NoSQL database that stores data in JSON format [64]. The database is organized into collections, with the "movies" collection being the primary collection that contains information about movies. The collection stores more than 23,000 documents that represent individual movies, with each document containing detailed information about the movie. The information stored includes the movie's title, release date, genre, rating, cast members, and other related information. The use of a document-oriented NoSQL database like Mflix ensures that our dataset is highly structured, this structured organization makes it easier to generate meaningful and precise queries corresponding to the given questions. The mflix database is an ideal dataset for creating MongoDB query-based datasets for answering questions. It offers a realistic representation of a movie streaming platform, with diverse movie information. The database's richness and relevancy ensure the capture of a wide range of features, improving the dataset's quality. Its depth and breadth of movie details allow for complex queries.



```
- {
    _id: ObjectId("573a1390f29313caabcd6377"),
    plot: `A rich young Easterner who has always wanted to live in "the Wild West" plans to move to
a Western town. Unknown to him, the town's "wild" days are long gone, and it is an orderly, ...`,
    genres: [ 'Comedy', 'Western', 'Romance' ],
    runtime: 72,
    cast: [
      'Douglas Fairbanks',
      'Eileen Percy',
      'Calvert Carter',
      'Charles Stevens'
    ],
    title: 'Wild and Woolly',
    fullplot: `A rich young Easterner who has always wanted to live in "the Wild West" plans to move
 to a Western town. Unknown to him, the town's "wild" days are long gone, and it is an orderly, civi
lized place now. The townsmen, not wanting to lose a rich potential resident, contrive to make over
the town to suit the young man's fantasy.`,
    languages: [ 'English' ],
    released: ISODate("1917-06-24T00:00:00.000Z"),
    directors: [ 'John Emerson' ],
    writers: [ 'Horace B. Carpenter (story)', 'John Emerson', 'Anita Loos' ],
    awards: { wins: 1, nominations: 0, text: '1 win.' },
    lastupdated: '2015-06-05 00:40:35.137000000',
    year: 1917,
    imdb: { rating: 6.9, votes: 388, id: 8775 },
    countries: [ 'USA' ],
    type: 'movie'
  }
-
```

FIGURE 3.1: Example of a Document from the "movies" collection

3.3.2 Description of MongoQpedia Dataset

The dataset used in our study, called **MongoQpedia**, (Mongo Query-pedia) is a collection of more than 80,000 question-answer pairs in natural language and their corresponding MongoDB queries. The questions were extracted from the Mflix movies database by identifying common types of questions that users might ask about movies. Initially, a set of templates was built as a starting point, which was later augmented using different data augmentation techniques like paraphrasing, back translation and named entity replacement (further information in Dataset construction section) to create a diverse and comprehensive set of questions. These questions cover various aspects of movies, such as release dates, genres, ratings, and cast members.

To obtain the MongoDB queries corresponding to each question-answer pair, we executed code that returns the desired answer. The annotation process was performed by a single annotator, who has experiences in the domain of movies and databases. The annotation process involved reviewing each question and identifying the corresponding MongoDB query that retrieves the answer from the Mflix movies database. The annotator employed a trial and error approach to generate the queries, testing the queries against the database to ensure their accuracy. The annotation process took approximately 300 hours to complete and validate the obtained dataset.

The dataset comprises a diverse set of questions and queries, including a variety of wh-questions such as "What is the release date of the movie?", "Who is the director of the movie?", and "How long is the movie?". The dataset also includes different types of MongoDB queries, such as find, count, distinct, and aggregate queries.
An example of the content of MongoQPedia dataset is presented in Table 3.1. It comprises a set of question-query pairs.

Question	MongoDB Query
What movie directed by Oliver Stone has the shortest running time?	db.movies.find({"directors":"Oliver Stone"}, {"title": 1, "runtime": 1}).sort({"runtime": 1}).limit(1)
what are the genres directed by Steven Spielberg?	db.movies.distinct("genres", {"directors": "Steven Spielberg" })
How many movies have the genre 'Action'?	db.movies.countDocuments({"genres": "Action"})
Who are the main actors in The Land Beyond the Sunset ?	db.movies.find({ "title": "The Land Beyond the Sunset" }, { "cast": 1})

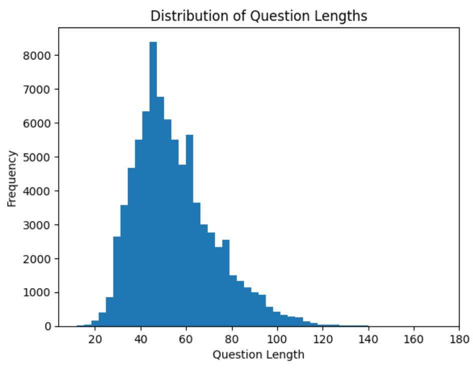
TABLE 3.1: Example of question-query pairs.

To evaluate the performance of machine learning models trained on the dataset, we split the data into three parts: a training set, a validation set, and a test set. The split used was 80/10/10, with 80% of the data used for training, 10% for validation, and 10% for testing. The split was randomized to ensure that the data in each set was representative of the entire dataset.

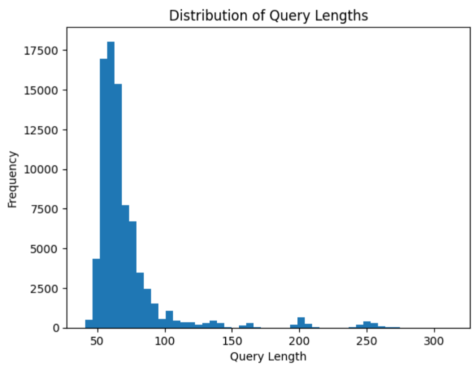
It is important to note that the queries are tailored specifically to the Mflix movies database and may not be generalizable to other databases or contexts. Despite the limitations, the dataset provides a valuable resource for studying natural language processing and database querying, particularly in the domain of movie-related information. Further augmentation of the dataset may be necessary to expand the variety of questions and queries in the dataset.

3.3.2.1 The MongoQpedia Dataset Statistics

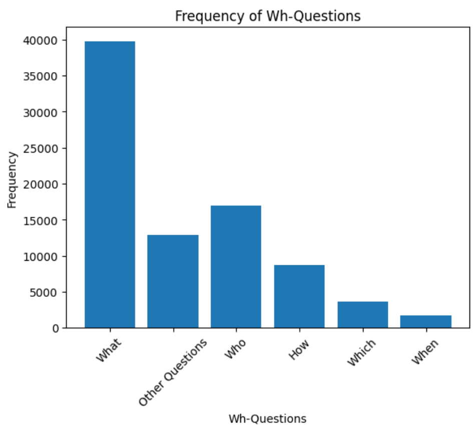
As stated earlier, our MongoQpedia dataset encompasses a diverse array of questions and queries. In order to enhance the comprehensiveness of our findings, we conducted an exploratory analysis of MongoQpedia dataset, including the distribution (Dist) of wh-question types (Figure 3.2c), MongoDB query types (Figure 3.2d), and statistics pertaining to query and natural language (NL) question lengths(Figure 3.2b , 3.2a).



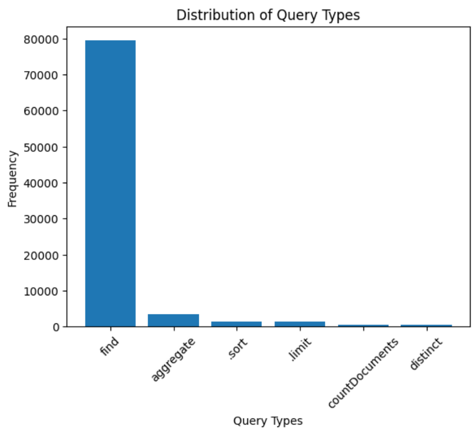
(A) Dist. of length of NL questions.



(B) Dist. of length of Mongodb queries.



(C) Dist. of types of NL questions.



(D) Dist. of types of Mongodb queries.

FIGURE 3.2: Distribution of questions and queries in MongoQpedia dataset.

In the next part, we present the detail of the process of MongoQpedia dataset-building.

3.3.3 The MongoQPedia Construction Process

In order to provide a comprehensive understanding of the dataset construction process, it is essential to first explore the intricacies of the data augmentation pipeline. This involves a closer examination of the specific techniques employed for natural language processing (NLP) data augmentation.

3.3.3.1 The Data Augmentation Pipeline

The data augmentation pipeline proposed for this study comprises four distinct steps, with steps 1, 2, and 4 involving the utilization of data augmentation techniques.

The first technique employed in the pipeline is paraphrasing (refer to "Paraphrasing" in Figure 3.4 and implementation in Algorithm 1). It involves leveraging a language model to rephrase sentences multiple times, thereby generating diverse question templates. Extensive research has demonstrated the effectiveness of this technique [65].

The second technique in the augmentation pipeline is back translation (refer to "Back Translation" in Figure 3.4 and implementation in Algorithm 2), is a widely used NLP data augmentation method known for its ability to produce high-quality outcomes. It entails translating a sentence from one language to another (or multiple languages) and then back to the original language (in this case, English). This process yields new templates that differ from the original ones. To facilitate this technique, the pipeline utilizes the Google Translate API, which is widely recognized for its reliable translation capabilities and extensive usage in various applications. Back translation has proven particularly valuable in scenarios where training resources for models are limited [66, 67, 68].

Finally, the last step of the pipeline introduces a technique called named entity replacement (refer to "Named Entity replacement" in Figure 3.4 and implementation in Algorithm 3), which operates at the sentence level. This technique involves modifying the templates by substituting the values of entities with others of the same type. For example, in a template containing the name of a director, the name is replaced with another director′s name extracted from a database. This technique can be considered an extension of some approaches mentioned in [69, 70], which further confirms the validity of this methodology.

3.3.3.2 The Construction Process

MongoQPedia is a closed-domain dataset comprising a large collection of movie-related questions and their corresponding MongoDB queries. To create this dataset, we used data augmentation techniques to generate a diverse set of high-quality questions and queries.

Creating a high-quality dataset for natural language processing requires a well-designed protocol for question creation and annotation. The one we adopt is as followed: The questions in the dataset should be simple, clear, and unambiguous, with no limit on their length. They should be written in a way that reflects natural human communication, and the dataset should be regularly reviewed and corrected to ensure semantic similarity between the questions. To be clear, Figure 3.3. shows a general overview of the Mongopedia Dataset creation workflow.

The first step in this process after the "Database Collection & Creation" was to create a set of human-annotated question templates. We created around 200 templates covering various aspects of movies, such as release dates, genres, ratings, and cast members. These templates were then augmented using three data augmentation methods: paraphrasing using a transformer model, back-translation, and named entity replacement.

Using the first two augmentation methods (see Algorithms 1 and 2), we were able to increase the number of templates to over 1000. However, the resulting templates were of varying quality and required cleaning. To ensure the quality of the dataset, we used a combination of automation scripts and human assistance to clean the templates, resulting in a final set of 700 high-quality templates. Figure 3.4 depicts a detail of the dataset augmentation techniques used with the validation step.

Although this process was time-consuming, it was crucial to ensure that the dataset is of high quality and can be effectively used for training machine learning models. The resulting dataset provides a valuable resource for developing and evaluating question-answering systems for movie-related queries.

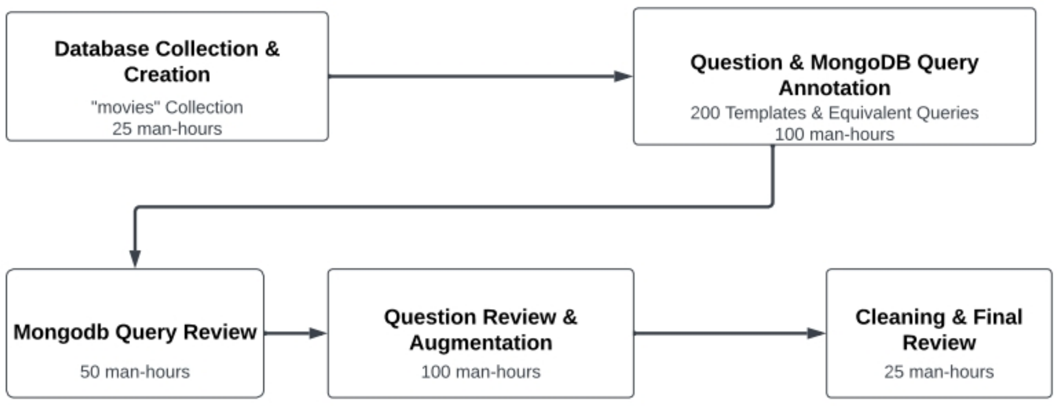

Database Collection & Creation
"movies" Collection
25 man-hours
Question & MongoDB Query Annotation
200 Templates & Equivalent Queries
100 man-hours
Mongodb Query Review
50 man-hours
Question Review & Augmentation
100 man-hours
Cleaning & Final Review
25 man-hours

FIGURE 3.3: Overview of the Mongopedia Dataset creation.

Algorithm 1 get_paraphrased_sentences function

```
   Input: A language model $model$, a tokenizer $tokenizer$, and a sentence $sentence$
   Output: A list of paraphrased sentences as strings
1: function GET_PARAPHRASED_SENTENCES($model, tokenizer, sentence$)
2:     $inputs \leftarrow$ tokenizer([$sentence$])    ▷ Tokenize the input sentence to get a list of token IDs
3:     $outputs \leftarrow$ model.generate(**inputs)    ▷ Generate the paraphrased sentences using the language model
4:     $paraphrased\_sentences \leftarrow$ tokenizer.batch_decode($outputs$)    ▷ Decode the generated sentences using the tokenizer to get them back to text
5:     return $paraphrased\_sentences$    ▷ Return the paraphrased sentences as a list of strings
6: end function
```

Algorithm 2 back_translation function

```
   Input: A template string $template$
   Output: A back translated string
1: function BACK_TRANSLATION($template$)
2:     $translator \leftarrow$ Translator($service\_urls = ['translate.google.com']$)    ▷ Create a translator object
3:     $translation \leftarrow$ translator.translate($template, src =' en', dest =' fr'$)    ▷ Translate the text from English to French
4:     $back\_translation \leftarrow$ translator.translate($translation.text, src =' fr', dest =' en'$)    ▷ Translate the French translation back to English
5:     return $back\_translation.text$    ▷ Return the back translated string
6: end function
```

To further increase the size of the MongoQPedia dataset, we used named entity replacement (see Algorithm 3). Using a Python script, we replaced entities in the question templates with similar ones that exist in the Mflix movies database. This process generated a dataset with more than 80,000 rows, significantly increasing the dataset's size.

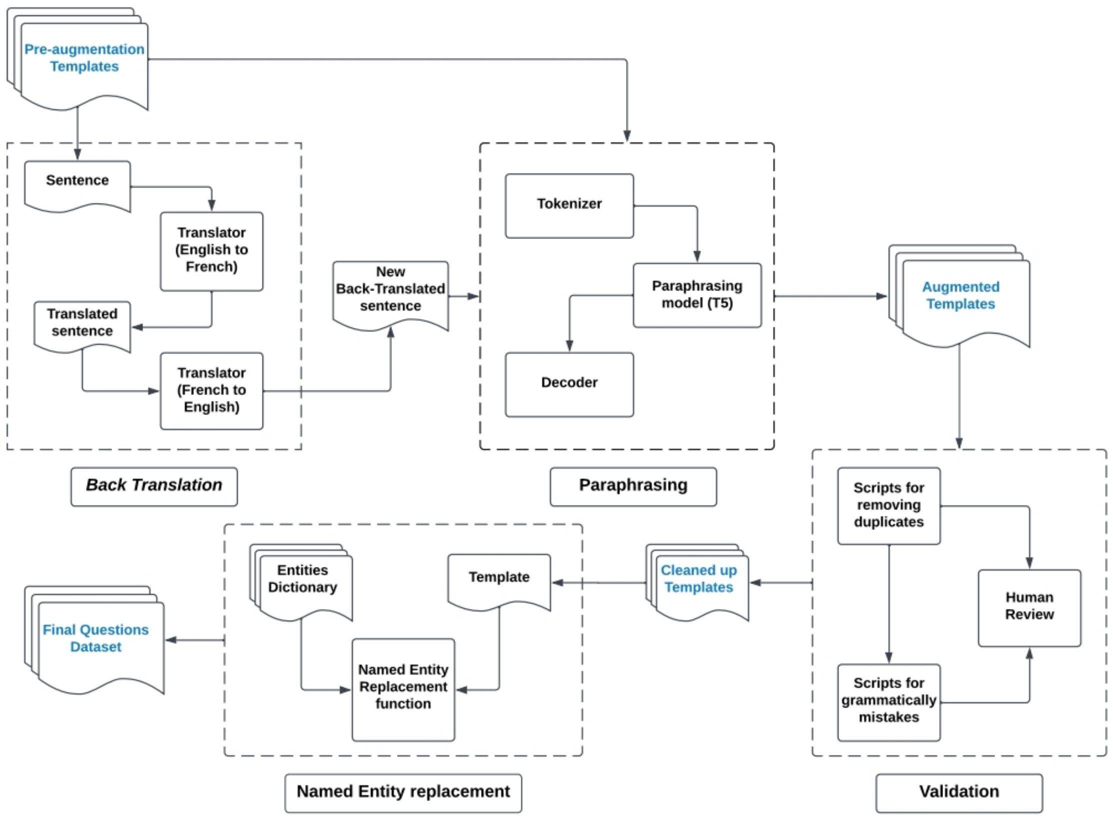


FIGURE 3.4: The Dataset Augmentation and Validation pipeline.

One of the benefits of using data augmentation techniques is that they can generate new data points that are similar but not identical to the original ones. This variability helps to reduce the risk of overfitting in machine learning models. In the case of MongoQPedia, the use of named entity replacement helped to further increase the dataset's size while preserving the semantic meaning of the questions.

The resulting dataset provides a valuable resource for training and evaluating machine learning models for question answering on MongoDB databases.

Algorithm 3 named_entity_replace function

```
    Input: A template string template with placeholders for named entities, and a
    dictionary named_entities mapping entity types to lists of entity names
    Output: A list of output templates with named entities replaced by their corresponding entity names
 1: function NAMED_ENTITY_REPLACE(template, named_entities)
 2:     Initialize an empty list output_templates
 3:     for each entity type entity_type in named_entities do
 4:         for each entity name entity_name in the list of entity names for entity_type do
 5:             Replace the named entity placeholder in template with the current entity_name, and store the result in replaced_template
 6:             Append replaced_template to the list of output templates output_templates
 7:         end for
 8:     end for
 9:     return output_templates
10: end function
```

Overall, the creation of MongoQPedia is a testament to the effectiveness of combining manual human effort with data augmentation techniques to produce high-quality datasets for natural language processing tasks, even when resources are limited. Our approach involved starting with a set of human-annotated question templates extracted from the Mflix movies database, followed by augmenting this set using paraphrasing and back-translation to generate a larger set of templates. Finally, named entity replacement was applied to further increase the dataset's size while preserving the semantic meaning of the questions.

This approach resulted in a dataset of more than 80,000 rows, which can be used for a variety of NLP tasks, including question answering and query generation. The importance of data cleaning and quality control was emphasized throughout the process to ensure the resulting dataset is of high quality and useful for machine learning tasks. Ultimately, the creation of MongoQPedia showcases the potential of data augmentation techniques to generate large, high-quality datasets and highlights the importance of combining human effort with automation in the dataset creation process.

3.4 Conceptual study of the proposed approach

Our Question Answering system based on the NoT5QL model for NoSQL databases using Transformer models falls under the category of a closed-domain factoid question-answering system for movie-related questions.

A closed-domain question-answering system is designed to operate within a specific domain or subject area. In this case, the subject area is movies. Closed-domain systems have the advantage of being more accurate and efficient than open-domain systems because they operate within a well-defined context.

A factoid question is a type of question that can be answered with a short, factual response. For example, "What is the release date of the movie 'Titanic'?" is a factoid question that can be answered with a specific date.

Our system is designed to generate MongoDB queries in response to factoid questions about movies. The queries generated by our system are specific to the movie-related domain and can be used to retrieve relevant data from a MongoDB database. By using Transformer models, our system can understand natural language queries and generate accurate MongoDB queries to answer factoid questions about movies.

3.4.1 Conceptual Architecture

Our proposed approach for building a question-answering system for NoSQL databases involves fine-tuning a pre-trained transformer model using our constructed dataset (see Figure 3.5). NoT5QL employs a pair of inputs: the task prefix (P) and the user question (Q). The task prefix guides the model by providing context for the query generation task. Initially, the input (P, Q) is encoded into a numerical representation that captures the meaning and context. This encoded representation is then used by the decoder component to generate the MongoDB Query (MQ), which corresponds to the user question.

Let's denote the task prefix as P and the user question as Q. To encode the input (P, Q) into a numerical representation, we can use a function E, which maps the input to the encoded representation.

The encoding function can be defined as:

$$Enc : (P, Q) \rightarrow \text{Encoded representation}$$

Once we have the encoded representation, it is used by the decoder component to generate the MongoDB Query (MQ). Let's denote the decoder function as D.

The decoder function can be defined as:

$$Dec : \text{Encoded representation} \rightarrow \text{MongoDB Query (MQ)}$$

Combining these elements, we can express the process of generating the MongoDB Query as follows:

$$\mathbf{MQ} = Dec(Enc(P, Q))$$

In this formula, $Enc(P, Q)$ represents the encoded representation of the input (P, Q), and $Dec(Enc(P, Q))$ represents the MongoDB Query generated by the decoder using the encoded representation.

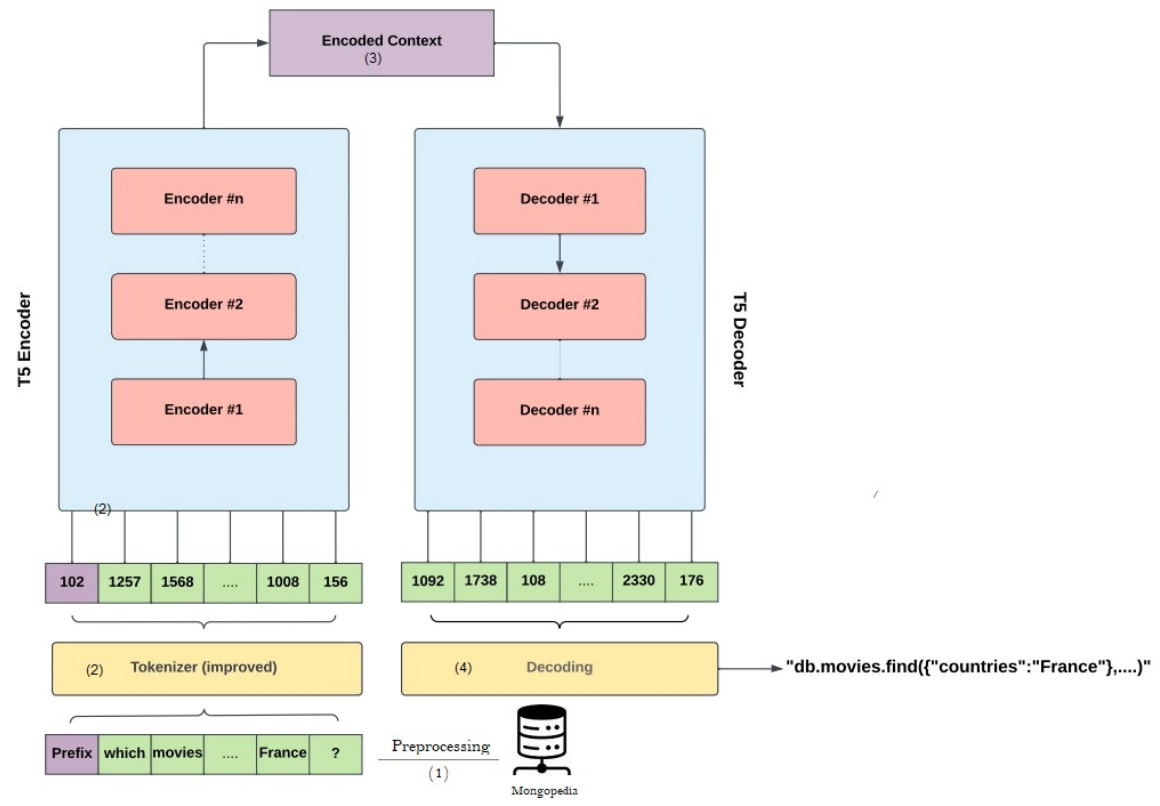


FIGURE 3.5: The architecture of the NoT5QL-based approach.

- Preprocessing: The preprocessing step in fine-tuning a T5 model involves introducing a task-specific prefix to provide context for the input sequence and specify the task being fine-tuned. For example, the prefix "Translate Question To MongoQuery:" is the one we picked for specific task. The input sequence is then tokenized and encoded before being fed into the T5 model for fine-tuning. The model learns to associate the task-specific prefix with the desired output for the given task, allowing it to generate accurate and relevant output for new input sequences.

- Improved Tokenizer Vocabulary: The T5 tokenizer is a powerful tool that can handle a wide variety of text inputs, but like any machine learning model, it has limitations. One of the limitations of the T5 tokenizer is that it encodes some code tokens, such as brackets and dollar signs, as unknown tokens or "unk" tokens. This can be a problem when working with specific programming languages, such as MongoDB, where these tokens are commonly used. To address this issue, our solution is to expand the vocabulary of the T5 tokenizer to include these code tokens. Once the vocabulary has been expanded, the T5 tokenizer can encode these code tokens correctly, without treating them as unknown tokens. This can be especially important when working with large datasets or complex codebases, where missing or incorrectly encoded code tokens can cause errors or inaccuracies in the analysis.

- Fine-Tuning: We fine-tune a pre-trained transformer model using the preprocessed dataset.
- Inference: We use the fine-tuned model to answer new questions related to NoSQL databases.

3.4.2 Pretrained Model

The pre-trained transformer model that we use for fine-tuning is the T5 model, which is a state-of-the-art transformer model for natural language processing (NLP) tasks [52]. T5 employs the transformer architecture introduced in [47], which includes extra decoder layers for sequence generation. T5 treats all NLP tasks as sequence to sequence generation and is pre-trained on a data-rich task before being fine-tuned for a specific downstream task.

The T5 model is capable of performing a wide range of NLP tasks, including text classification, machine translation, and question answering. The model achieves state-of-the-art performance on several benchmark NLP datasets.

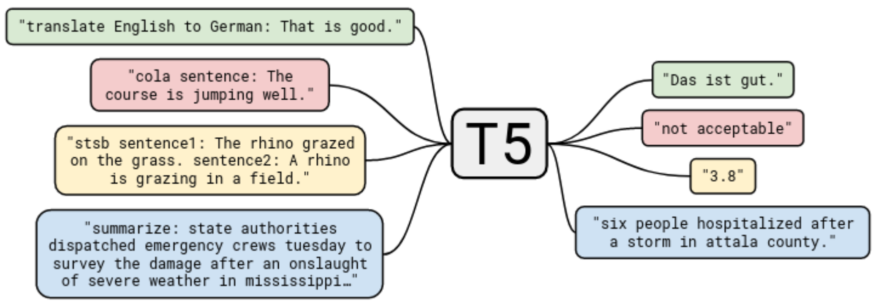

"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

FIGURE 3.6: A diagram of T5's text-to-text framework. Every task we consider is cast as feeding our model text as input and training it to generate some target text [52].

3.5 Conclusion

This chapter presented our conceptual study for developing a dynamic QA system for the MongoDB database.

The aim of this study is twofold: (1) The dataset building and (2) the proposition of deep learning model.

We created a new dataset called "MongoQPedia" of movies that we used specifically to train and evaluate our model. The creation of the "MongoQPedia" dataset represents an important contribution to the community. By providing a new dataset that is specifically designed for training and evaluating NoSQL question-answering systems, we are enabling researchers and practitioners to better understand the strengths and limitations of different approaches and to develop new and more effective models in the future.

In addition, our work represents an important contribution to two important fields; natural language processing (NLP) and database. By developing a model that can effectively answer questions about NoSQL databases, we have opened up new possibilities for applications in areas such as data analytics, machine learning, and artificial intelligence.

Overall, our work represents a significant step forward in the development of natural language processing systems for NoSQL databases, and we believe that it will have a positive impact on a wide range of applications and industries.

To validate the performance of our proposal, the experimental study will be the goal of the next chapter.

CHAPTER 4 IMPLEMENTATION AND EXPERIMENTATION

4.1 Introduction

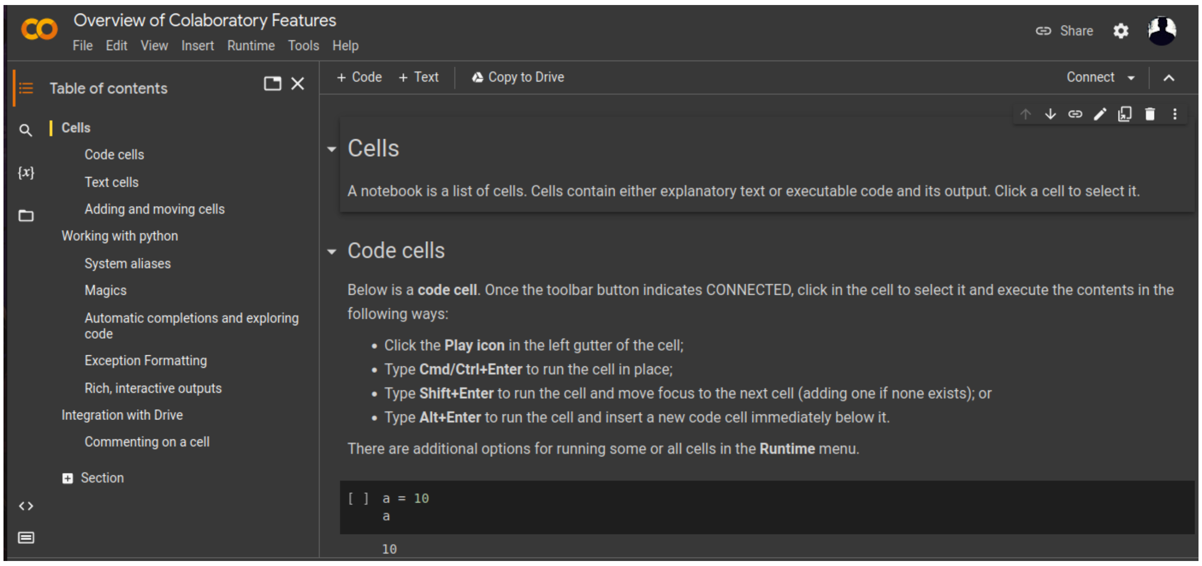
In the previous chapter, we discussed the conceptual study of our Q&A system for MongoDB. In this chapter, we will delve into the implementation details and performance evaluation. The chapter is divided into three main sections. The first section focuses on the development environment, which includes the hardware and software requirements necessary for setting up the system. The second section provides an overview of the libraries used in the implementation process. Finally, the third section explains the main parts of the implementation and provides a detailed analysis of the results obtained from our Q&A system and how to do model inference. By the end of this chapter, readers will have a clear understanding of the development process and the various tools used in the implementation of our Q&A system.

4.2 Development environment

4.2.1 Google Colaboratory

Colaboratory, or "Colab" for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing access free of charge to computing resources including GPUs [71]. Colab also provides more utilities such as:

- Integration with Google Drive: Google Colab is integrated with Google Drive, which makes it easy to store and share notebooks and datasets.
- Pre-installed Libraries: Google Colab comes pre-installed with many popular Python libraries, such as NumPy, Pandas, and Matplotlib. This makes it easy to get started with data analysis and machine learning without having to install and configure these libraries manually.
- Pre-installed Deep Learning Frameworks: Google Colab comes pre-installed with popular deep learning frameworks like TensorFlow, Keras, PyTorch, and fastai. This saves users a significant amount of time and effort that would otherwise be spent installing and configuring these frameworks.
- High RAM and Disk Space: Google Colab provides users with access to high RAM and disk space, which can be essential for training large deep learning models that require significant amounts of memory.
- Easy Access to Pre-trained Models: Google Colab provides easy access to pre-trained deep learning models that can be used for transfer learning. This can save significant amounts of time and resources in training deep learning models from scratch.



Cells

A notebook is a list of cells. Cells contain either explanatory text or executable code and its output. Click a cell to select it.

Code cells

Below is a **code cell**. Once the toolbar button indicates CONNECTED, click in the cell to select it and execute the contents in the following ways:

- Click the **Play icon** in the left gutter of the cell;
- Type **Cmd/Ctrl+Enter** to run the cell in place;
- Type **Shift+Enter** to run the cell and move focus to the next cell (adding one if none exists); or
- Type **Alt+Enter** to run the cell and insert a new code cell immediately below it.

There are additional options for running some or all cells in the **Runtime** menu.

```
[ ]  a = 10
     a

     10
```

FIGURE 4.1: Colab Interface

4.2.2 Docker

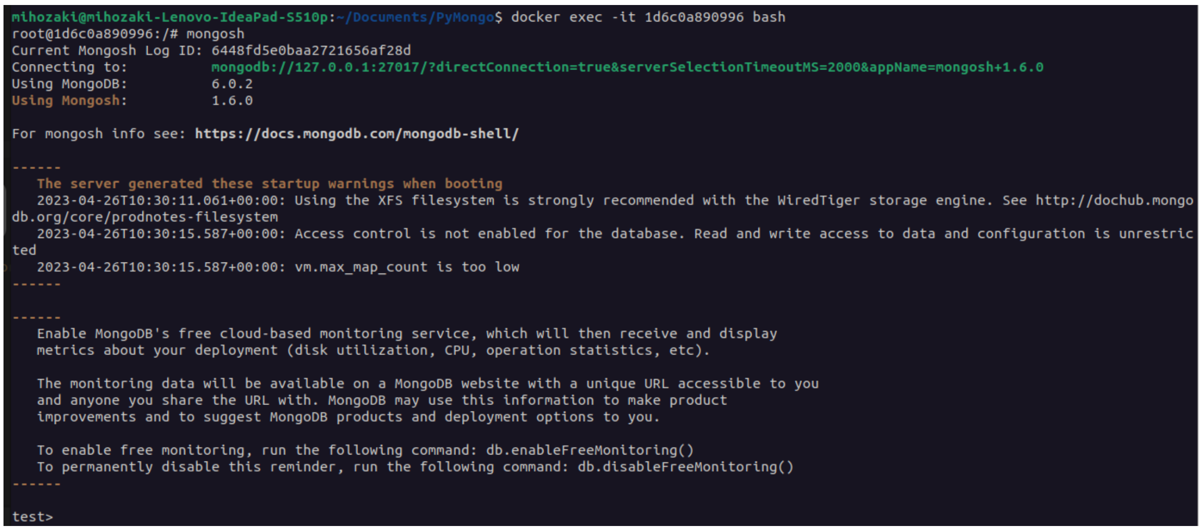
Docker is an open platform for developing, shipping, and running applications [72]. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allows you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host. You can easily share containers while you work, and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.

- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.



```
mihozaki@mihozaki-Lenovo-IdeaPad-S510p:~/Documents/PyMongo$ docker exec -it 1d6c0a890996 bash
root@1d6c0a890996:/# mongosh
Current Mongosh Log ID: 6448fd5e0baa2721656af28d
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1.6.0
Using MongoDB:          6.0.2
Using Mongosh:          1.6.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2023-04-26T10:30:11.061+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongo
db.org/core/prodnotes-filesystem
   2023-04-26T10:30:15.587+00:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestric
ted
   2023-04-26T10:30:15.587+00:00: vm.max_map_count is too low
------

------
   Enable MongoDB's free cloud-based monitoring service, which will then receive and display
   metrics about your deployment (disk utilization, CPU, operation statistics, etc).

   The monitoring data will be available on a MongoDB website with a unique URL accessible to you
   and anyone you share the URL with. MongoDB may use this information to make product
   improvements and to suggest MongoDB products and deployment options to you.

   To enable free monitoring, run the following command: db.enableFreeMonitoring()
   To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
------

test>
```

FIGURE 4.2: Docker Command Line Interface

4.2.3 Hugging Face

HuggingFace is an AI community that promotes open source contributions [73]. It is a hub of open source models for Natural Language Processing, computer vision, and other fields where AI plays its role. Even the tech giants like Google, Facebook, AWS, Microsoft, and others use the models, datasets, and libraries.HuggingFace provides state-of-the-art models for different tasks. It has a vast number of pre-trained models for different tasks. At the time of writing this article (August 2022), there were more than 61k pre-trained models. Following are the tasks supported by HuggingFace:

- NLP tasks: HuggingFace is famous for its contribution to the NLP domain. The NLP tasks are: Text classification, Text generation, Translation, Summarization, Fill-mask, Question-Answering, Zero-shot classification, Sentence similarity.

- Computer vision tasks: The computer vision tasks are as follows: Image classification, Image segmentation, Object detection.
- Audio tasks: The audio tasks are as follows: Speech recognition, Text-to-speech, Automatic Speech recognition, Audio classification

In HuggingFace, the Transformers library, allows us to use these models in a way that abstracts unnecessary details.

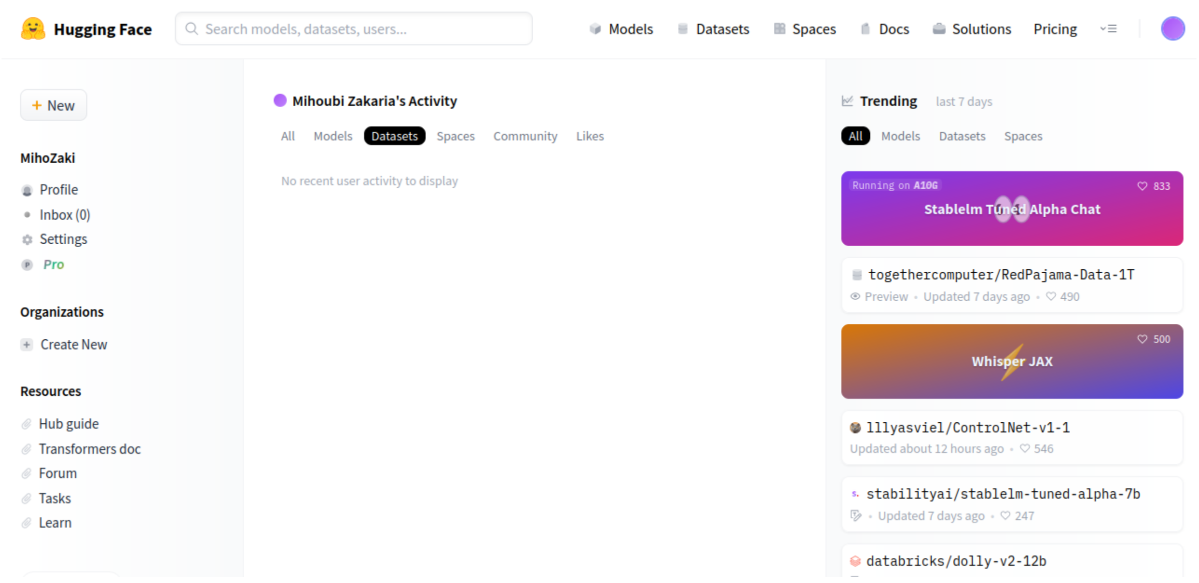


FIGURE 4.3: Hugging Face Interface

4.3 Programming Languages and Libraries Used

4.3.1 Programming Languages:

- **Python:** Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms [74].

- **Mongodb Query Language (MQL):** MQL, or MongoDB Query Language, is a powerful and flexible query language that is specifically designed for querying and manipulating data in MongoDB databases. MQL is a declarative language, meaning that users specify what they want the database to do rather than how to do it. This allows users to write complex queries and manipulate data without needing to have an in-depth understanding of the underlying database structure or implementation.

4.3.2 Libraries Used:

- **The Natural Language Toolkit (NLTK):** is a platform used for building Python programs that work with human language data for applying in statistical natural language processing (NLP) [75]. It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning . It also includes graphical demonstrations and sample data sets as well as accompanied by a cook book and a book which explains the principles behind the underlying language processing tasks that NLTK supports.

- **Transformer:** Transformers provides APIs and tools to easily download and train state-of-the-art pretrained models [76]. Using pretrained models can reduce your compute costs, carbon footprint, and save you the time and resources required to train a model from scratch. These models support common tasks in different modalities, such as:

  - **Natural Language Processing:** text classification, named entity recognition, question answering, language modeling, summarization, translation, multiple choice, and text generation.
  - **Computer Vision:** image classification, object detection, and segmentation.
  - **Audio:** automatic speech recognition and audio classification.

  - **Multimodal:** table question answering, optical character recognition, information extraction from scanned documents, video classification, and visual question answering.

4.4 Experimentation and Results

In this section, we will explore the process used to fine-tune the model for the task of Answering Natural Language Questions and discuss the results achieved.

4.4.1 The Fine Tuning Process

In this subsection, we describe the experimental setup for fine-tuning the T5 model on the "MongoQpedia" dataset. We discuss the loading of dataset splits, T5 checkpoint, and tokenizer. We also outline the preprocessing steps performed on the dataset and the setup of hyperparameters. Finally, we explain the fine-tuning process and the evaluation metrics used.

- **Dataset and T5 Model Loading:** We begin by loading the three dataset splits of "MongoQpedia": train, validation, and test. Each split contains a set of questions and corresponding queries. Additionally, we load the T5 checkpoint and its respective tokenizer.

- **Preprocessing and Tokenization** To prepare the dataset for fine-tuning, we apply specific preprocessing steps. Firstly, we set the maximum input length to 256 and the maximum target length to 512, considering the limitations of the T5 model. Next, we utilize the "preprocess_data" function, which preprocesses the dataset by adding the new prefix "Translate Question To MongoQuery:" to the elements of the "Questions" column. These elements are considered the "Source Text." We then tokenize the source text and tokenize the elements of the "Queries" column, which serve as the target text. Finally, we return the model inputs required for fine-tuning.

- **Hyper-parameter Setup** In this step, we set up the hyper-parameters for the fine-tuning process. We define the learning rate, batch size, and the number of training epochs. These hyper-parameters play a crucial role in optimizing the performance of the fine-tuned model. we experimented with multiple configurations for both the "small" and "base", which are showed in the table 4.1 where the highlighted ones are the one picked as the final configuration:

TABLE 4.1: Fine tuning hyper-paramaters Configurations

Model Version	Batch Size	Learning Rate	Number of Epochs
T5-Small	16	4e-5	4
T5-Small	16	3e-4	6
T5-Small	8	1e-4	2
T5-Base	8	4e-5	4
T5-Base	16	3e-4	6
T5-Base	8	1e-4	2

- **Fine-tuning Process** With the dataset preprocessed and the hyperparameters set, we initiate the fine-tuning process. We train the T5 model using the training dataset split and perform updates based on the selected hyperparameters. The model is iteratively fine-tuned over multiple epochs to improve its performance on the given task.

4.4.2 Evaluation and Results

In this section, we will share the results of the fine tuning process (Model accuracy and Model loss) for T5-Base then evaluate it using the BLEU and ROUGE Metrics and compare it to T5-Small.

4.4.2.1 Model Accuracy

Model accuracy is a measure of how well the translation model performs in terms of correctly predicting the target translations. It is typically calculated by comparing the predicted translations generated by the model with the reference translations (ground truth). The accuracy is determined by the percentage of correct predictions out of the total number of predictions. Figure 4.4 illustrate the accuracy of the chosen model (base version).

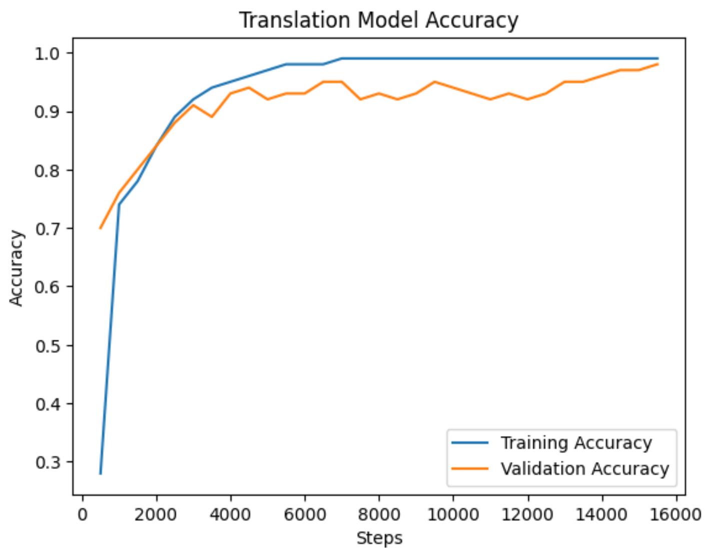


FIGURE 4.4: Model Accuracy (T5-Base)

4.4.2.2 Model Loss

Model loss, also known as training loss or objective loss, is a quantitative measure of the dissimilarity between the predicted translations and the ground truth translations. It represents how well the model is able to minimize the difference between its predictions and the desired outputs during the training process. Figure 4.5 illustrate the Loss of the chosen model (base version).

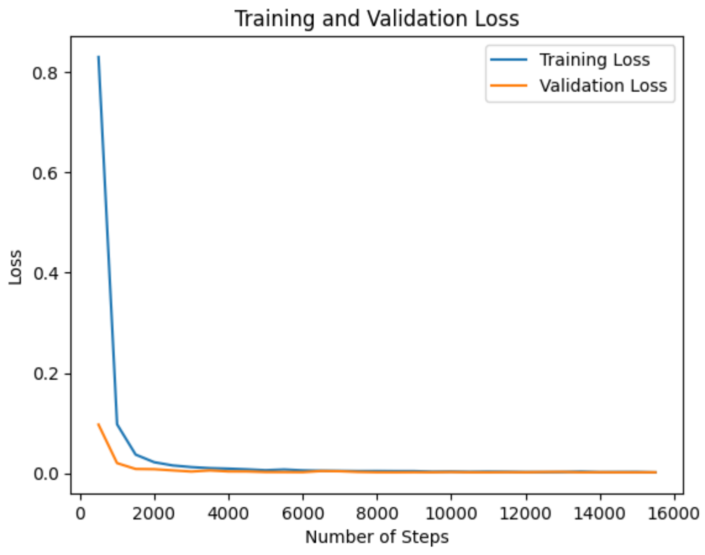


FIGURE 4.5: Model Loss (T5-Base)

4.4.2.3 Evaluation

While model accuracy and loss are useful for monitoring the performance of The model during training, they do not provide a complete picture of the translation quality. To gain a more comprehensive understanding, it is important to perform a final evaluation using a separate test dataset and metrics such as BLEU (Bilingual Evaluation Understudy) and ROUGE (Recall-Oriented Understudy for Gisting Evaluation). By evaluating our models with a test dataset and using metrics like BLEU and ROUGE, we can obtain a more objective assessment of its performance. These metrics consider factors such as the coverage of important content, word order, and overall similarity to human translations.

- **BLEU (Bilingual Evaluation Understudy)**: BLEU [77]is a widely used metric for evaluating the quality of machine translation systems. It measures the degree of overlap between the generated output and a set of reference translations based on n-grams (contiguous sequences of words).

BLEU score formula:

$$\text{BLEU} = \text{BP} \times \exp\left(\sum_{n=1}^{N} w_n \log \text{P}_n\right)$$

where:

$$\text{BP} = \begin{cases} 1, & \text{if } c > r \\ \exp(1 - \frac{r}{c}), & \text{if } c \leq r \end{cases}$$
$$\text{P}_n = \frac{\sum_{\text{clip} \in \text{clip}_n} \text{clip}}{\sum_{\text{cand} \in \text{cand}_n} \text{cand}}$$

The metric computes a geometric mean of the n-gram precisions, where the n-gram precision measures the percentage of n-grams in the generated output that appear in the reference translations. BLEU ranges from 0 to 1, where a higher score indicates better performance. BLEU has been shown to have a strong correlation with human judgments of translation quality.

- **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)**: ROUGE [78] is a set of metrics for evaluating the quality of summarization systems. It measures the overlap between the generated summary and a set of reference summaries based on n-grams and word sequences.

$$\text{ROUGE} = \frac{\text{ROUGE-N}}{m} + \frac{\text{ROUGE-L}}{m} + \frac{\text{ROUGE-S}}{m}$$

where:

$$\text{ROUGE-N} = \frac{\sum_{\text{reference summaries}} \sum_{\text{n-grams}} \text{count\_match}(n\text{-gram})}{\sum_{\text{reference summaries}} \sum_{\text{n-grams}} \text{count}(n\text{-gram})}$$

$$\text{ROUGE-L} = \frac{\sum_{\text{reference summaries}} \text{longest\_common\_subsequence}}{\sum_{\text{reference summaries}} \text{summary length}}$$

$$\text{ROUGE-S} = \frac{\sum_{\text{reference summaries}} \sum_{\text{skip-bigrams}} \text{count\_match}(\text{skip-bigram})}{\sum_{\text{reference summaries}} \sum_{\text{skip-bigrams}} \text{count}(\text{skip-bigram})}$$

ROUGE includes several variants, such as ROUGE-N (based on n-grams), ROUGE-L (based on longest common subsequence), and ROUGE-W (based on weighted longest common subsequence). ROUGE ranges between 0 and 1, where a higher score indicates better performance. ROUGE has been shown to have a high correlation with human judgments of summarization and translation quality.

Due to the absence of base-line model for this specific task, we opted to compare both of the T5-Base and T5-Small perfromances. The table 4.2 shows the evaluation results of T5-Small and T5-Base models.

TABLE 4.2: Comparison of BLEU and ROUGE Scores for T5-Base and T5-Small

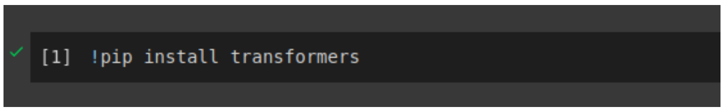
Metric	**T5-Small**	**T5-Base**
BLEU	0.89	0.97
ROUGE	0.90	0.98

The T5-base model's larger architecture and increased parameter count allowed it to capture more complex query patterns and better understand the structure and syntax of MongoDB queries. As a result, it exhibited a higher accuracy and efficiency in generating correct and optimized queries for various question types. The T5-base model's enhanced capacity enabled it to handle a wider range of query scenarios, making it more adept at providing accurate and relevant answers to questions related to MongoDB. Its superior performance in this specific task demonstrates the advantage of having a larger and more powerful model for handling complex database query tasks.

4.4.3 Model Inference

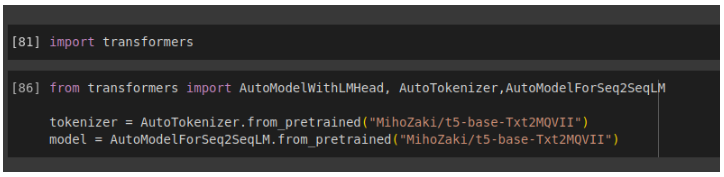
Machine learning (ML) inference is the process of feeding live data points into a machine learning algorithm (or "ML model") to generate an output such as a single numerical score. This is also known as "operationalizing an ML model" or "putting an ML model into production." [79]. To perform "inference" on our model you can follow next steps:

1. Install the required libraries:



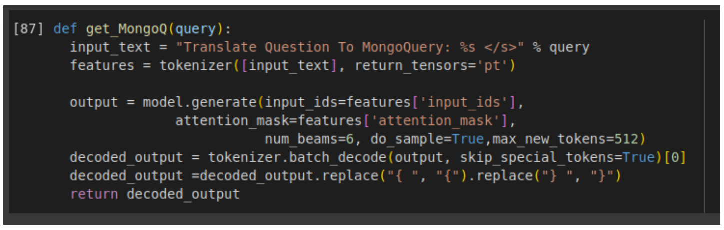
```
[1]  !pip install transformers
```

2. Load the model and tokenizer: Import the necessary modules and Load the specified Model.



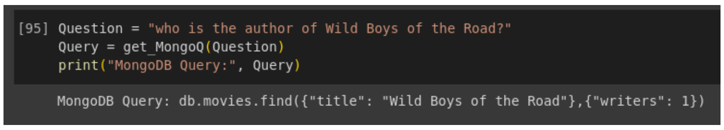
```
[81] import transformers

[86] from transformers import AutoModelWithLMHead, AutoTokenizer,AutoModelForSeq2SeqLM

     tokenizer = AutoTokenizer.from_pretrained("MihoZaki/t5-base-Txt2MQVII")
     model = AutoModelForSeq2SeqLM.from_pretrained("MihoZaki/t5-base-Txt2MQVII")
```

3. Define the function for generating answers:



```
[87] def get_MongoQ(query):
       input_text = "Translate Question To MongoQuery: %s </s>" % query
       features = tokenizer([input_text], return_tensors='pt')

       output = model.generate(input_ids=features['input_ids'],
                    attention_mask=features['attention_mask'],
                           num_beams=6, do_sample=True,max_new_tokens=512)
       decoded_output = tokenizer.batch_decode(output, skip_special_tokens=True)[0]
       decoded_output =decoded_output.replace("{ ", "{").replace("} ", "}")
       return decoded_output
```

4. Use the model for inference: Now The user can ask questions and receive the answer in type of a MongoDB Query.



```
[95] Question = "who is the author of Wild Boys of the Road?"
     Query = get_MongoQ(Question)
     print("MongoDB Query:", Query)

     MongoDB Query: db.movies.find({"title": "Wild Boys of the Road"},{"writers": 1})
```

4.5 Conclusion

In conclusion, this chapter presented an experiment focused on fine-tuning T5-base and T5-Small models on our proprietary dataset, "MongoQpedia" for the task of answering Questions with mongodb queries. The objective was to explore different hyperparameter configurations and evaluate their performance using BLUE and ROUGE metrics.

The results of the experiment revealed that T5-base outperformed T5-Small by a significant margin of ∼8% . This performance difference suggests that the larger T5-base model has a greater capacity to capture and generate more accurate and contextually relevant responses compared to T5-Small.

The utilization of T5 models in our approach yielded satisfactory results overall. The fine-tuning process allowed us to tailor the models to our specific dataset, enhancing their understanding of the context and enabling them to generate more relevant answers. This demonstrates the effectiveness of T5 models in handling our specific domain and dataset.

The use of BLUE and ROUGE metrics provided a quantitative assessment of the model's performance. These metrics are widely recognized in the natural language processing community and offer valuable insights into the quality of the generated responses. By employing these evaluation measures, we were able to objectively compare the performance of the two models and make informed conclusions.

In summary, our experiment demonstrated the superiority of T5-base over T5-Small for our task of question answering on the MongoQpedia dataset. The fine-tuning process, combined with the appropriate selection of hyperparameters, played a crucial role in achieving these results. The satisfactory outcomes obtained through our approach highlight the potential of T5 models in addressing similar challenges in the field of natural language processing.

GENERAL CONCLUSION

Question-answering systems are a prominent area of research in the field of natural language processing (NLP) and artificial intelligence (AI). They aim to bridge the gap between human language and machine understanding by enabling computers to comprehend and respond to questions posed in natural language. Traditionally, question-answering systems were designed to operate on structured and curated data. However, with the rise of NoSQL databases, such as MongoDB, that store data in a more flexible and unstructured format, the need for question answering systems tailored specifically for these databases, has emerged.

In this thesis, we explored the effectiveness of question answering systems for NoSQL databases, specifically MongoDB, by leveraging the power of the T5 model, where we proposed the NoT5QL model and carefully created a dataset named MongoQpedia. By employing techniques such as paraphrasing, back translation, and named entity replacement, the dataset was augmented to improve the model's understanding and performance.

The fine-tuning process of the T5 model, utilizing both the small and base variations, demonstrated promising outcomes. Our experiments revealed that the base model outperformed the small model, achieving satisfactory results in terms of accuracy and effectiveness in answering questions posed to MongoDB.

The augmentation techniques employed in the creation of MongoQpedia proved to be valuable in enhancing the model's comprehension and generating diverse and accurate responses. Paraphrasing, back translation, and named entity replacement enriched the dataset, enabling the model to generalize well and provide more robust answers.

These findings highlight the potential of leveraging NoT5QL-based question answering systems for NoSQL databases, specifically MongoDB. The success of the base model indicates that larger models may further improve performance, warranting future exploration. Furthermore, this research opens up avenues for further investigations into other augmentation techniques and the application of NoT5QL-based models for other NoSQL databases.

In our future works we aim for the following perspectives:

- **Dataset Expansion**: Continuously expand and update MongoQpedia with new Domain-related questions (sports, history, science, ...) and their corresponding MongoDB queries. This will help improve the coverage and diversity of the dataset.

- **Integration with MongoDB**: Explore the possibility of integrating the T5 model with MongoDB itself, enabling direct querying from within the database. This would require developing a suitable interface and ensuring compatibility between your model and MongoDB's query language.

- **Investigate Mongodb Injection**: Research the possibility of generating MongoDB Query injection and increasing the security of the system.

- **The Creation of a new Mongodb Specific Tokenizer**: The creation of a new tokenizer that's specialized in Mongodb query language, which can speed up the inference.

- **Chat-like system**: exploring the possibility of implementing a chat system that can handle multi-parts questions.

- **Query Optimization**: MongoDB query performance can be affected by various factors such as indexing, query structure, and data distribution. we can investigate techniques to optimize generated queries to improve their efficiency and execution time.
- **Query Validation and error handling**: Design our model to handle cases where the generated query may be syntactically or semantically incorrect. we can integrate query validation mechanisms to ensure that the generated queries are valid and handle potential errors gracefully.
- **Expanding the Data Augmentation pipeline**: Research more about NLP augmentation techniques and drawing inspiration from image processing techniques.

BIBLIOGRAPHY

[1] A. B. M. Moniruzzaman and Syed Akhter Hossain. "NoSQL database: New era of databases for big data analytics-classification, characteristics and comparison". In: *arXiv preprint arXiv:1307.0191* (2013).

[2] Abdelghani Bouziane et al. "Question answering systems: survey and trends". In: *Procedia Computer Science* 73 (2015), pp. 366–375.

[3] Gourav Bathla, Pardeep Singh, and Rahul K Singh. "AI-based Question Answering system for NoSQL standard query". In: (2021).

[4] Robert T Mason. "NoSQL databases and data modeling techniques for a document-oriented NoSQL database". In: *Proceedings of Informing Science & IT Education Conference (InSITE)*. Vol. 3. 4. 2015, pp. 259–268.

[5] Ameya Nayak, Anil Poriya, and Dikshay Poojary. "Type of NOSQL databases and its comparison with relational databases". In: *International Journal of Applied Information Systems* 5.4 (2013), pp. 16–19.

[6] Andreas Meier and Michael Kaufmann. *SQL & NoSQL databases*. Springer, 2019.

[7] S George. "NoSQL–NOT ONLY SQL". In: *International Journal of Enterprise Computing and Business Systems* 2.2 (2013).

[8] Strozzi C. *NoSQL: a non SQL RDBMS*.

[9] Vatika Sharma and Meenu Dave. "Sql and nosql databases". In: *International Journal of Advanced Research in Computer Science and Software Engineering* 2.8 (2012).

[10] Antonios Makris et al. "A classification of NoSQL data stores based on key design characteristics". In: *Procedia Computer Science* 97 (2016), pp. 94–103.

[11] Steve Ataky Tsham Mpinda, Luıs Gustavo Maschietto, and Patrick Andjasubu Bungama. "From relational database to column-oriented nosql database: migration process". In: *International Journal of Engineering Research & Technology (IJERT)* 4 (2015), pp. 399–403.

[12] Aicha Aggoune and Mohamed Sofiane Namoune. "Metadata-driven Data Migration from Object-relational Database to NoSQL Document-oriented Database". In: *Computer Science* 23.4 (2022), pp. 495–519.

[13] Deka Ganesh Chandra. "BASE analysis of NoSQL database". In: *Future Generation Computer Systems* 52 (2015), pp. 13–21.

[14] Armando Fox and Eric A Brewer. "Harvest, yield, and scalable tolerant systems". In: *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*. IEEE. 1999, pp. 174–178.

[15] Jing Han et al. "Survey on NoSQL database". In: *2011 6th international conference on pervasive computing and applications*. IEEE. 2011, pp. 363–366.

[16] *What is the CAP Theorem?* URL: `https://hazelcast.com/glossary/cap-theorem/`. (accessed: 06.02.2023).

[17] Seth Gilbert and Nancy Lynch. "Perspectives on the CAP Theorem". In: *Computer* 45.2 (2012), pp. 30–36.

[18] Pranabjyoti Bordoloi. *ACID, CAP, and BASE*. URL: `https://medium.com/@pranabj.aec/acid-cap-and-base-cc73dee43f8c`. (accessed: 01.02.2023).

[19] Kolade Chris. *CRUD Operations – What is CRUD?* URL: `https://www.freecodecamp.org/news/crud-operations-explained/`. (accessed: 02-02-2023).

[20] Arfan Sharif. *What is CRUD?* URL: https://www.crowdstrike.com/cybersecurity-101/observability/crud/. (accessed: 02-02-2023).

[21] Franck Michel, Catherine Faron-Zucker, and Johan Montagnat. "A mapping-based method to query MongoDB documents with SPARQL". In: *Database and Expert Systems Applications: 27th International Conference, DEXA 2016, Porto, Portugal, September 5-8, 2016, Proceedings, Part II 27*. Springer. 2016, pp. 52–67.

[22] Houcine Matallah, Ghalem Belalem, and Karim Bouamrane. "Comparative study between the MySQL relational database and the MongoDB NoSQL database". In: *International Journal of Software Science and Computational Intelligence (IJSSCI)* 13.3 (2021), pp. 38–63.

[23] Alexandru Boicea, Florin Radulescu, and Laura Ioana Agapin. "MongoDB vs Oracle–database comparison". In: *2012 third international conference on emerging intelligent data and web technologies*. IEEE. 2012, pp. 330–335.

[24] MongoDB. *MongoDB CRUD Operations*. URL: https://www.mongodb.com/docs/v4.2/crud/. (accessed: 08-02-2023).

[25] Jeff Carpenter and Eben Hewitt. *Cassandra: The Definitive Guide,(Revised)*. " O'Reilly Media, Inc.", 2022.

[26] CassandraOFF. *Cassandra Crud Operation – Create, Update, Read & Delete last accessed*. URL: https://data-flair.training/blogs/cassandra-crud-operation/. (accessed: 08-02-2023).

[27] Shannon Kempe. *UnQL: A Standardized Query Language for NoSQL Databases*. URL: https://www.dataversity.net/unql-a-standardized-query-language-for-nosql-databases/. (accessed: 09-02-2023).

[28] Peter Buneman, Mary Fernandez, and Dan Suciu. "UnQL: a query language and algebra for semistructured data based on structural recursion". In: *The VLDB Journal* 9 (2000), pp. 76–110.

[29] Nadime Francis et al. “Cypher: An evolving query language for property graphs”. In: *Proceedings of the 2018 international conference on management of data*. 2018, pp. 1433–1445.

[30] Neo4jOF. *Neo4j CheatSheet*. URL: `https://neo4j.com/docs/cypher-cheat-sheet/current/#_set`. (accessed: 08-02-2023).

[31] MongoDB. *NoSQL vs. SQL Databases*. URL: `https://www.mongodb.com/nosql-explained/nosql-vs-sql`. (accessed: 13.01.2023).

[32] Keith D. Foote. *NoSQL Databases: Advantages and Disadvantages*. URL: `https://www.dataversity.net/nosql-databases-advantages-and-disadvantages/`. (accessed: 13.01.2023).

[33] Sanjay K Dwivedi and Vaishali Singh. “Research and reviews in question answering system”. In: *Procedia Technology* 10 (2013), pp. 417–424.

[34] Ali Mohamed Nabil Allam and Mohamed Hassan Haggag. “The question answering systems: A survey”. In: *International Journal of Research and Reviews in Information Sciences (IJRRIS)* 2.3 (2012).

[35] Bolanle Ojokoh and Emmanuel Adebisi. “A review of question answering systems”. In: *Journal of Web Engineering* 17.8 (2018), pp. 717–758.

[36] Ellen M Voorhees and Donna K Harman. “The text retrieval conference”. In: *TREC: Experiment and evaluation in information retrieval* (2005), pp. 3–19.

[37] Jonathan Berant et al. “Semantic parsing on freebase from question-answer pairs”. In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1533–1544.

[38] Amit Mishra and Sanjay Kumar Jain. “A survey on question answering systems with classification”. In: *Journal of King Saud University-Computer and Information Sciences* 28.3 (2016), pp. 345–361.

[39] Tahseen Sultana and Srinivasu Badugu. "A review on different question answering system approaches". In: *Advances in Decision Sciences, Image Processing, Security and Computer Vision: International Conference on Emerging Trends in Engineering (ICETE), Vol. 2*. Springer. 2020, pp. 579–586.

[40] Nitin Indurkhya and Fred J Damerau. *Handbook of natural language processing*. Chapman and Hall/CRC, 2010.

[41] Oleksandr Kolomiyets and Marie-Francine Moens. "A survey on question answering technology from an information retrieval perspective". In: *Information Sciences* 181.24 (2011), pp. 5412–5434.

[42] Hang Cui, Min-Yen Kan, and Tat-Seng Chua. "Soft pattern matching models for definitional question answering". In: *ACM Transactions on Information Systems (TOIS)* 25.2 (2007), 8–es.

[43] Ryuichiro Higashinaka and Hideki Isozaki. "Corpus-based question answering for why-questions". In: *Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-I*. 2008.

[44] Suzan Verberne et al. "Discourse-based answering of why-questions". In: *Proceedings of the Workshop on Frontiers in Linguistically Annotated Corpora 2007: Balancing Richness and Availability*. 2007, pp. 37–40.

[45] Suzan Verberne et al. "Using syntactic information for improving why-question answering". In: *Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008)*. 2008, pp. 1182–1186.

[46] Suzan Verberne et al. "What is not in the Bag of Words for Why-QA?" In: *Computational Linguistics* 36.2 (2010), pp. 229–245.

[47] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[48] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[49] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach”. In: *arXiv preprint arXiv:1907.11692* (2019).

[50] Victor Sanh et al. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. In: *arXiv preprint arXiv:1910.01108* (2019).

[51] Pengcheng He et al. “Deberta: Decoding-enhanced bert with disentangled attention”. In: *arXiv preprint arXiv:2006.03654* (2020).

[52] Colin Raffel et al. “Exploring the limits of transfer learning with a unified text-to-text transformer”. In: *The Journal of Machine Learning Research* 21.1 (2020), pp. 5485–5551.

[53] Alec Radford et al. “Language models are unsupervised multitask learners”. In: *OpenAI blog* 1.8 (2019), p. 9.

[54] Mike Lewis et al. “Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension”. In: *arXiv preprint arXiv:1910.13461* (2019).

[55] Karam Ahkouk et al. “SQLSketch: Generating SQL Queries using a sketch-based approach”. In: *Journal of Intelligent & Fuzzy Systems* 40.6 (2021), pp. 12253–12263.

[56] Rami Reddy, Nandi Reddy, and Sivaji Bandyopadhyay. “Dialogue based question answering system in telugu”. In: *Proceedings of the Workshop on Multilingual Question Answering-MLQA ‘06*. 2006.

[57] Ping Wang, Tian Shi, and Chandan K Reddy. “Text-to-SQL generation for question answering on electronic medical records”. In: *Proceedings of The Web Conference 2020*. 2020, pp. 350–361.

[58] Victor Zhong, Caiming Xiong, and Richard Socher. “Seq2sql: Generating structured queries from natural language using reinforcement learning”. In: *arXiv preprint arXiv:1709.00103* (2017).

[59] Xiaojun Xu, Chang Liu, and Dawn Song. "Sqlnet: Generating structured queries from natural language without reinforcement learning". In: *arXiv preprint arXiv:1711.04436* (2017).

[60] Sebastian Blank et al. "Querying NoSQL with deep learning to answer natural language questions". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 9416–9421.

[61] Phuc Do, Truong HV Phan, and Brij B Gupta. "Developing a Vietnamese tourism question answering system using knowledge graph and deep learning". In: *Transactions on Asian and Low-Resource Language Information Processing* 20.5 (2021), pp. 1–18.

[62] T Pradeep, PC Rafeeque, and Reena Murali. "Natural Language To NoSQL Query Conversion using Deep Learning". In: ().

[63] Mahesh D Gadekar et al. "Natural Language (English) To MongoDB Interface". In: *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)* 4.3 (2015).

[64] MongoDBOFF. *sample mflix*. URL: `https://www.mongodb.com/docs/atlas/sample-data/sample-mflix/`. (accessed: 15.01.2023).

[65] Emre Okur, Shubhangi Sahay, and Liza Nachman. "Data augmentation with paraphrase generation and entity extraction for multimodal dialogue system". In: *arXiv preprint arXiv:2205.04006* (2022).

[66] Rico Sennrich, Barry Haddow, and Alexandra Birch. "Improving neural machine translation models with monolingual data". In: *arXiv preprint arXiv:1511.06709* (2015).

[67] Marzieh Fadaee, Arianna Bisazza, and Christof Monz. "Data augmentation for low-resource neural machine translation". In: *arXiv preprint arXiv:1705.00440* (2017).

[68] Usama Yaseen and Stefan Langer. “Data augmentation for low-resource named entity recognition using backtranslation”. In: *arXiv preprint arXiv:2108.11703* (2021).

[69] Dinesh Rajagopal et al. “Counterfactual data augmentation improves factuality of abstractive summarization”. In: *arXiv preprint arXiv:2205.12416* (2022).

[70] Xiang Dai and Heike Adel. “An analysis of simple data augmentation for named entity recognition”. In: *arXiv preprint arXiv:2010.11683* (2020).

[71] Google Colab. *colaboratory Faq*. URL: `https://research.google.com/colaboratory/faq.html`. (accessed: 25.04.2023).

[72] Docker. *Get Started*. URL: `https://docs.docker.com/get-started/overview/`. (accessed: 25.04.2023).

[73] Hugging Face. *What is HuggingFace?* URL: `https://www.educative.io/answers/what-is-huggingface`. (accessed: 25.04.2023).

[74] python. *The Python Tutorial?* URL: `https://docs.python.org/3/tutorial/`. (accessed: 25.04.2023).

[75] Margaret Rouse. *What Does Natural Language Toolkit Mean?* URL: `https://www.techopedia.com/definition/30343/natural-language-toolkit-nltk`. (accessed: 25.04.2023).

[76] Hugging Face. *transformers Docs*. URL: `https://huggingface.co/docs/transformers/index`. (accessed: 25.04.2023).

[77] Kishore Papineni et al. “Bleu: a method for automatic evaluation of machine translation”. In: *Proceedings of the 40th annual meeting on association for computational linguistics*. 2002, pp. 311–318.

[78] Chin-Yew Lin. “Rouge: A package for automatic evaluation of summaries”. In: *Text summarization branches out*. 2004, pp. 74–81.

[79] Hazelcast. *What Is Machine Learning Inference?* `https://hazelcast.com/glossary/machine-learning-inference/`.