

Experiments with text-to-SPARQL based on ChatGPT

Caio Viktor S. Avila*, Vânia M.P.Vidal*, Wellington Franco* and Marco A. Casanova†

*Departament of Computing, Federal University of Ceará

Fortaleza, Brazil 60440-900

Email: caioviktor@alu.ufc.br, vvidal@lia.ufc.br, wellington@crateus.ufc.br

†Department of Informatics, Pontifical Catholic University of Rio de Janeiro

Rio de Janeiro, Brazil 22451-900

Email: casanova@inf.puc-rio.br

Abstract—Currently, large language models (LLMs) are the state of the art for pre-trained language models. LLMs have been applied to many tasks, including question and answering over Knowledge Graphs (KGs) and text-to-SPARQL, that is, the translation of Natural Language questions to SPARQL queries. With such motivation, this paper first describes preliminary experiments to evaluate the ability of ChatGPT to answer NL questions over KGs. Based on these experiments, the paper introduces Auto-KGQAGPT, an autonomous domain-independent framework based on LLMs for text-to-SPARQL. The framework selects fragments of the KG, which the LLM uses to translate the user’s NL question to a SPARQL query on the KG. Finally, the paper describes preliminary experiments with Auto-KGQAGPT with ChatGPT that indicate that the framework substantially reduced the number of tokens passed to ChatGPT without sacrificing performance.

I. INTRODUCTION

Question Answering (QA) systems are computational systems that can answer questions typically asked in Linguagem Natural (LN) [1]. Among the types of QA systems, *Knowledge Graph Question Answering* (KGQA) systems stand out for their ability to generate curated and deep responses. KGQA systems are QA systems that are based on a *Knowledge Graph* (KG). These systems translate NL questions into formal queries in the language the KG supports, such as SPARQL [2].

Several architectures and approaches have been proposed for KGQA systems, but those based on Deep Learning models became popular in recent years [3]. Currently, Large Language Models (LLMs) are the state of the art for pre-trained language models. LLMs achieve good results in a large number of tasks, including question and answering over KGs and text-to-SPARQL, that is, the translation of Natural Language questions to SPARQL queries [4], which opens an opportunity to develop KGQA systems based on LLMs [5].

With such motivation, this paper first describes preliminary experiments to evaluate the ability of ChatGPT (with GPT-3.5) to answer NL questions over a KG. The experiments cover a scenario where ChatGPT receives an NL question and the KG as input and direct generates a response, and scenarios where ChatGPT first translates an NL question to a SPARQL query which is then executed on the KG, receiving as input the T-Box (schema) and A-Box (instances) of the KG, both

individually and jointly. These experiments were carried out manually on a toy KG in the domain of families, people, and jobs, accompanied by comments describing the facts encoded in NL.

Based on these preliminary experiments, the paper introduces Auto-KGQAGPT, an autonomous domain-independent framework based on LLMs for text-to-SPARQL. Given a KG K , the framework selects fragments of the T-Box and A-Box of K , which the LLM uses to translate the user’s original NL question Q to a SPARQL query on K . Auto-KGQAGPT generates n SPARQL queries that translate Q and selects the best one, based on the query results on K . Finally, Auto-KGQAGPT generates an NL response to the user, based on Q , the selected SPARQL query S , and the result of the execution of S over K . The main goal of Auto-KGQA is the selection of smaller KG fragments thereby reducing the number of tokens passed as input to the LLM, which decreases the possibility of hallucinations with irrelevant elements in the input and increases the ability to support large KGs.

The main contributions of this paper, therefore, are: (1) A dataset with simple and complex questions to evaluate text-to-SPARQL tools; (2) Preliminary experiments with the use of ChatGPT as a text-to-SPARQL tool; (3) A standalone framework, based on LLMs, for text-to-SPARQL that selects KG fragments to reduce the number of tokens in the input.

The remainder of this article is structured as follows. Section II covers related work. Section III describes preliminary experiments using ChatGPT to answer NL questions over a KG. Section IV introduces Auto-KGQAGPT. Section V presents the results and discussions about the experiments with ChatGPT and Auto-KGQAGPT. Finally, Section VI contains the conclusions.

II. RELATED WORK

An approach to translate NL queries to SPARQL, called SGPT, is proposed in [6]. SGPT encodes a vector of linguistic embedding features, such as parts-of-speech (POS) and the dependency tree, in addition to the corresponding sub-graph of the question. Embeddings are generated using stacks of Transformers. Then, the embeddings are passed as training to a Transformers model based on GPT-2 to generate the SPARQL

query. The model was evaluated on LC-QuAD 2.0, VQuAnDA and QALD-9, and is still state of the art in the latter.

FoodGPT, introduced in [7], is an LLM targeting the food testing domain. *FoodGPT* was incrementally trained using structured data sources and scanned documents. An external KG was used as a consultation source, reducing the hallucination phenomenon. When using KG, *FoodGPT* employs a retrieval model to analyze the user query and retrieve relevant knowledge. This knowledge is combined with the query and provided as input to *FoodGPT*, generating responses based on the graph fragment and the knowledge acquired in training.

The *Knowledge Solver* (KSL) paradigm was proposed in [8] to teach LLMs to fetch essential knowledge from external knowledge bases. In this approach, the LLM receives as input question-and-answer pairs (of the multiple choice type), where the LLM task is to learn to select the KG subgraph required to answer the original NL question. To achieve this, the authors developed a prompt strategy to transform retrievals into a sequence of multi-hop decisions. The approach was tested in three different LLMs (GPT-3.5, LLaMA, and LLaMA-2) in three different datasets (*CommonsenseQA*, *OpenbookQA*, and *MedQA-USMLE*), achieving significant improvements in accuracy in the first two datasets in a zero-shot configuration, and even better results when the model was fine-tuned.

SPARQLGEN is a one-shot generative approach to create SPARQL queries by augmenting LLMs with the relevant context within a single prompt [9]. The proposed prompt is composed of the question to be translated, an RDF subgraph required to translate the query, and an example of a correct SPARQL query for a different question. The approach used GPT-3 to test on 3 different datasets QALD-9, QALD-10 and BESTIARY (a dataset built by the authors themselves). In the approach used, the selection of subgraphs passed as input in the prompt for each question was carried out using a reverse engineering process in the expected correct SPARQL queries (ground truth query), constructing the subgraph from the basic graph patterns (BGP)¹ present in the query, replacing variables by their values returned when executing the SPARQL query, projecting all the variables involved, thus generating the minimum subgraph to answer the question. As a result, the authors found that passing the underlying KG and a random query example improves the generation results.

III. USING CHATGPT TO ANSWER NL QUESTIONS

This section describes preliminary experiments to evaluate the ability of ChatGPT (with GPT-3.5) to answer NL questions over a KG. The evaluation consists of 4 manual experiments, described in Sections III-B, III-C and III-D, using ChatGPT (with GPT-3.5). The textual logs of the conversation carried out with ChatGPT in these experiments can be found at the link². Section III-E summarizes the lesson learned, which guided the development of Auto-KGQAGPT, presented in Section IV.

¹https://www.w3.org/2001/sw/DataAccess/rq23/sparql-defns.html#defn_BasicGraphPattern

²<https://bit.ly/41qfjGM>

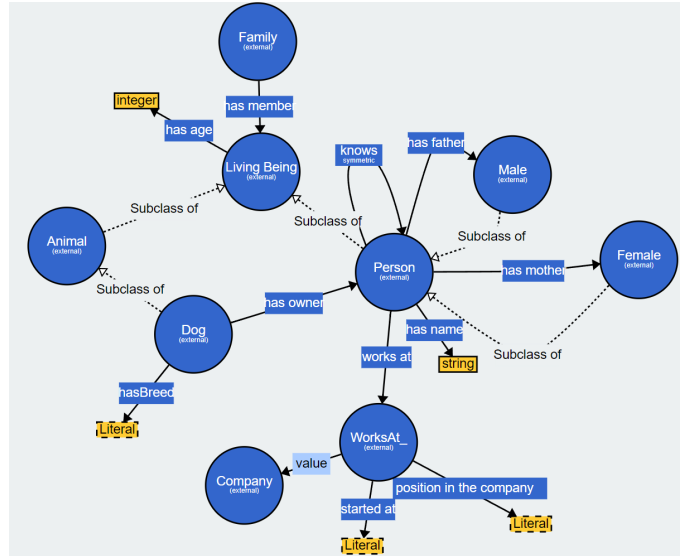


Fig. 1. Scheme of the KG used in the experiments

A. A Toy KG and Sample NL Questions

The evaluation is based on a toy KG in RDF, whose schema is shown in Figure 1. The KG has a simplified definition of the classes Living Beings, People, Animals, Companies, Employment, etc. The RDF serialization of the KG is available at the link³.

The evaluation used 15 NL questions, defined over the KG, which test ChatGPT’s ability to respond to simple queries that can be answered based on a single triple, and complex queries that require using relationship paths or aggregation functions.

Table I contains the NL questions, where “Id” identifies the NL question, “Question” is the text of the NL question, and “Justification” briefly justifies why the NL question was included. Additional information about each NL question, including the SPARQL query translation and the result set that act as gold-standard, can be found at the link⁴.

B. Direct Answering of NL Questions

This experiment investigated ChatGPT’s ability to answer NL questions on a KG, directly passed as input, without performing queries on external sources. Therefore, in this scenario, ChatGPT has to interpret the input KG, understand the NL question, and answer it based on the knowledge learned directly from the graph.

The input KG was passed to ChatGPT just once, at the beginning of the interaction, in Turtle RDF [10]. Each NL question was passed within a single chat instance isolated from other conversations, and structured as the following prompt:

Taking into account the graph, {question}

The response evaluation was done manually by comparing the response text with the response expected by the gold-standard.

³<https://bit.ly/3GJvvt2>

⁴<https://bit.ly/3RFCLwj>

TABLE I
QUESTIONS USED IN THE EXPERIMENTS

Id	Question	Justification
Q ₁	who is John?	Simple query to evaluate whether the tool can correctly associate a query term with a resource. This makes a group formed by queries 1,2,3 to evaluate whether the tool produces variations of the same query pattern correctly
Q ₂	who is Tom?	...
Q ₃	who is Buddy?	...
Q ₄	How old is John Doe?	Simple query that looks for a specific attribute of a specific resource
Q ₅	who John Doe Knows?	Simple query that fetches triples for a specific relationship from a specific resource
Q ₆	who is the oldest person?	Aggregation query that uses a term not present directly in the KG schema, requiring correct interpretation
Q ₇	who is the youngest person?	...
Q ₈	how many persons exist?	...
Q ₉	what is the average age of people?	...
Q ₁₀	who is over 30?	Query using filter on specific attribute, but the attribute is implicit
Q ₁₁	what is the relation between Jane and Tom?	Query seeking to identify the relationship between two specific resources
Q ₁₂	Does John Doe have a job? if so, where does he work, his position and when did he start?	Complex query composed of multiple simple queries, also involving the need to abstract a class representing a relationship
Q ₁₃	who is the father of buddy's owner?	complex query that relies on identifying intermediate resources using a relationship to a specific resource as a filter
Q ₁₄	how old is the father of the person with 25 years old?	complex query that depends on identifying intermediate resources using a specific attribute as a filter
Q ₁₅	who is Barack Obama?	query that should not be answered by the graph

C. Experiments with text-to-SPARQL using the A-Box and the T-Box

This experiment tests ChatGPT’s ability to translate NL questions into SPARQL queries on a given KG passed as input.

The input KG was again passed to ChatGPT just once, at the beginning of the interaction, in Turtle RDF. Each NL question was again passed within a single chat instance isolated from other conversations and structured as the following prompt:

```
Write a SPARQL query for the question using only
classes and properties defined in the graph, and
remember to include the base URI declaration in
the query and non-essential properties as
optional:
{question}
```

The response evaluation was done manually by extracting the SPARQL query from the text returned by ChatGPT and executing it on the complete KG, stored in a triplestore with inferences enabled. At the end of the query execution, the result set was compared with that expected by the gold-standard.

D. Experiments text-to-SPARQL using only the T-Box or the A-box

This section describes two experiments, where the T-Box and A-Box were separately passed to ChatGPT in Turtle RDF format at the beginning of each conversation. Each NL question was again passed within a single Chat instance isolated from other conversations and structured as the prompt:

```
Write a SPARQL query for the question using only
classes and properties defined in the graph, and
remember to include the base URI declaration in
the query and non-essential properties as
optional if needed. Declare filters on strings
```

```
as filter operations over regex function using
the case insensitive flag.
The question is the following:
{question}
```

The response evaluation was done as for Section III-C.

E. An Analysis of the Preliminary Experiments

The results of the experiments of Sections III-B, III-C and III-D suggest that the text-to-SPARQL approach, passing both the T-Box and the A-Box, generated the best results, considering the limitations of ChatGPT (with GPT-3.5).

However, when trying to replicate this process to larger KBs, the limit on the number of tokens allowed by the OpenAI API⁵ proved to be a challenge. Consequently, Section IV introduces an approach for selecting the KG fragments (subgraphs) most relevant for an NL question, considering both the T-Box and the A-Box.

IV. AUTO-KGQAGPT: AN LLM-BASED FRAMEWORK FOR TEXT-TO-SPARQL

Given the results of the preliminary experiments, reported in Section III-E, this section proposes Auto-KGQAGPT, an autonomous domain-independent framework based on LLMs for text-to-SPARQL. The discussion that follows uses ChatGPT as the LLM, but the framework is otherwise generic.

A. Outline of the Framework

The proposed framework receives as input an NL question about a KG and returns as output a response also in NL, generated based on the result of executing, on the KG, the SPARQL query translated from the original NL question.

⁵<https://platform.openai.com/docs/guides/text-generation/chat-completions-api>

Algorithm 1: GenerateIndexes

Input: endpoint**Output:** selected_resources

```
1 Function GenerateIndexes(endpoint):
  // Create T-Box index
2   t_box_terms ← endpoint.retrieveTBox();
3   t_box_index ← newIndex();
4   t_box_index ← PopulateIndex(t_box_index,
  t_box_terms, endpoint);
  // Create A-Box index
5   a_box_terms ← endpoint.retrieveABox();
6   a_box_index ← newIndex();
7   a_box_index ← PopulateIndex(a_box_index,
  a_box_terms, endpoint);
8   return t_box_index, a_box_index;
```

The framework workflow is divided into two stages – the offline stage and the online stage.

The offline stage receives the KG (T-Box and A-Box) as input via a SPARQL endpoint and creates two indexes, which are used in the online stage. The first index generates mappings from labels of classes and properties to their URIs, thus acting as a mapping of NL terms to T-Box resources. The second index generates mappings from A-Box (instances) resource labels to their URIs, thus acting as a mapping of NL terms to A-Box resources.

The online stage receives an NL question S as input, translates S into a SPARQL query Q using the LLM, and executes Q on the KG to generate the response sent to the user. This stage uses the indices generated in the offline stage to identify references to KG terms in the NL question. The LLM receives only fragments of the KG related to these terms.

B. Offline Stage

GenerateIndexes Algorithm

Algorithm 1 creates indices for the terms occurring in the T-Box (Lines 2-4) or the A-Box (Lines 5-7). It receives as input the endpoint where the KG is loaded.

Function `endpoint.retrieveTBox()` (Line 2) retrieves the classes and properties of the KG, where each line of the response will have the URI of a term (class or property), its label (if it exists) and the type of the term. The algorithm creates a new object for the T-Box index (Line 3) and populates it using the `PopulateIndex()` function defined in Algorithm 2.

A similar process is followed to create the A-Box index, except that it uses the `endpoint.retrieveABox()` function.

PopulateIndex Algorithm

Algorithm 2 receives as input a list of terms T and generates an index for T as output.

For each term, the algorithm decides the search key to be used (Line 9), preferably the explicitly defined label of the term (Line 4), or a label inferred from its URI (Line 7). The current version of the index uses a full-text search mechanism, similar to that used in *elasticsearch* [11], where the entry will

Algorithm 2: PopulateIndex

Input: index, terms, endpoint**Output:** index

```
1 Function PopulateIndex(index, terms, endpoint):
2   for each term in terms do
3     if term.label ≠ "" then
4       label ← term.label;
5     end
6     else
7       label ← inferLabelFromURI(term.uri);
8     end
9     indexKey ← normalize_input(label);
10    term_in_triples_count ←
  endpoint.countTriples(term.uri);
11    index.add(indexKey, {'uri': term.uri, 'label':
  term.label, 'type': term.type,
  'triples_participates': term_in_triples_count});
12  end
13  return index;
```

be compared as fragments of the index keys. Each element of the index has attributes such as the URI of the term, the original label, its type (*class, property or resource*) and the number of triples in which the term appears in the KG (*triples_participate* obtained in Line 10). The text contained in the search key is normalized (Line 9), where possible terms in camel case notation are decoded. Then, the text is converted to lowercase, and finally separated into tokens, which in turn will be replaced by their lemmatized versions.

C. Online Stage

AnswerQuestion Algorithm

Algorithm 3 receives as input the NL question, the endpoint where the KG is loaded, the indices generated offline, the maximum number of edge hops, and the maximum number of edges per property (explained in what follows).

The algorithm first normalizes the input (Line 2), using the same process as that applied to the search keys in the offline stage. The result is a sequence of lemmatized tokens that represents the original input.

Then, the algorithm retrieves the possible KG terms referenced in the input NL question (Lines 3 and 5), described in more detail in Algorithm 4. It uses these terms to select KG fragments (lines 4 and 6) from the T-Box and the A-Box, concatenates the fragments, and converts them to Turtle RDF serialization (Line 7), generating a single subgraph, which is passed to the LLM to be used as a vocabulary of five SPARQL queries that aims to translate the NL question (Line 9).

The algorithm executes the SPARQL query variations (Lines 12 to 17) and uses their results to select which one best represents the original intent of the NL question (Line 19).

At the end of the process, the algorithm uses the LLM to generate the final NL response (Line 23), using as a basis the original NL question, the selected SPARQL query, and its response set.

Algorithm 3: AnswerQuestion Algorithm

Input: question, endpoint, index_t_box, index_a_box, edge_hops, max_edges_by_property
Output: answer

- 1 visited_nodes \leftarrow set();
- 2 normalized_question \leftarrow normalize_input(question);
- 3 t_box_resources \leftarrow SelectResources(normalized_question, index_t_box);
- 4 t_box_triples \leftarrow SelectTriples(endpoint, visited_nodes, t_box_resources, visited_nodes, n_hops=2);
- 5 a_box_resources \leftarrow SelectResources(normalized_question, index_a_box);
- 6 a_box_triples \leftarrow SelectTriples(endpoint, visited_nodes, a_box_resources, visited_nodes, n_hops=2);
- 7 triples \leftarrow convert_to_turtle(t_box_triples + a_box_triples);
- 8 prompt_translation \leftarrow generate_prompt_translation(question, triples);
- 9 answers \leftarrow chatGPT(prompt_translation, n=5);
- 10 queries \leftarrow [];
- 11 results \leftarrow [];
- 12 **for** answer **in** answers **do**
- 13 | sparql \leftarrow extractSPARQL(answer);
- 14 | result \leftarrow endpoint.run_sparql(sparql);
- 15 | queries.add(sparql);
- 16 | results.add(result);
- 17 **end**
- 18 prompt_best_query \leftarrow generate_prompt_best_query(question, queries, results);
- 19 best_query_index \leftarrow extract_option_best_query(chatGPT(prompt_best_query));
- 20 best_sparql \leftarrow queries[best_query_index];
- 21 best_result \leftarrow results[best_query_index];
- 22 prompt_answer \leftarrow generate_prompt_final_answer(question, best_sparql, best_result);
- 23 answer \leftarrow chatGPT(prompt_answer, temperature=1.2);
- 24 **return** answer;

SelectResources Algorithm

Algorithm 4 identifies references to KG elements by searching the sequence of normalized tokens of the input question to find possible references to KG terms.

First, the *GetPossibleMatches()* function (Line 1) separates the tokens into n-grams, and searches for them in the T-Box or A-Box indexes created in the offline step. The creation of candidate n-grams is performed through a sliding window [12] pattern matching process of decreasing size.

For each n-gram found in an index, the algorithm chooses only one candidate term in the KG as follows. It first computes the relevance scores of the candidate terms (Line 6). This score considers the similarity value between the search key

Algorithm 4: SelectResources

Input: normalized_question, index
Output: selected_resources

- 1 matches \leftarrow GetPossibleMatches(normalized_question, index);
- 2 selected_resources \leftarrow [];
- 3 **for** match **in** matches **do**
- 4 | scores \leftarrow [];
- 5 | **for** hit **in** match.hits_index **do**
- 6 | | scores[hit] \leftarrow (hit.similarity_to_input + hit.triples_participates) / 2;
- 7 | **end**
- 8 | selected_hit \leftarrow argmax(scores);
- 9 | resource_uri \leftarrow selected_hit.uri;
- 10 | selected_resources.add(resource_uri);
- 11 **end**
- 12 **return** selected_resources;

strings and the n-gram in question, and the number of triples in which the resource appears in the KG. These values are already normalized to a scale between 0 and 1, and their simple average is computed, resulting in a term relevance between 0 and 1. The algorithm selects the candidate term with the highest relevance (Line 8), and returns its URI (Line 10).

Lastly, the algorithm returns the list of URIs of the relevant terms for the input question (Line 12).

SelectTriples Algorithm

Algorithm 5 selects the triples that define the KG fragment.

For each relevant term t that Algorithm 4 returns, the algorithm retrieves from the KG all triples in which the URI of t appears as subject or object (Line 5). It, therefore, captures all properties to and from a node. However, triples that do not add relevant semantic value to the resource are filtered out. Some examples of excluded triples are those that define the term as a *rdfs:Resource* or *owl:Thing*, triples that indicate self-references resulting from the application of axioms, such as triples that indicate that a class is a subclass (*rdfs:subClassOf*) of itself, a class is equivalent (*owl:equivalentClass*) to itself, or a resource is the same (*owl:sameAs*) as itself.

In addition to the triples directly related to a term t , the algorithm also captures the triples related to the neighbors of t and the neighbors of the neighbors of t , in a recursive process (Line 8), up to a depth defined by the “edge_hops” parameter.

Prompt Patterns

The prompt template (line 9 from Algorithm 3) that requests the generation of SPARQL queries that are translation variations of the original NL question is as follows:

System:

Consider the following RDF graph written in Turtle syntax:

```
{KG_Subgraph}
```

Write a SPARQL query for the question given by the user following the restrictions:

```

-Use only classes and properties defined in the
RDF graph, for this is important to use the same
URIs for the properties and classes as defined
in the original graph;
-Include all the prefixes used in the SPARQL
query;
-Declare non-essential properties to the
question as OPTIONAL if needed;
-DO NOT use specific resources in the query;
Declare filters on strings (like labels and
names) as filter operations over the REGEX
function using the case-insensitive flag.
User:
{Question}

```

The prompt template starts with the KG subgraph obtained from the fragments of terms relevant to the NL question. Then, it contains a list of restrictions to be followed to generate correct queries according to the KG, which are generic with respect to the given examples. The prompt template ends with the user’s NL question.

The prompt template (line 19 from Algorithm 3) that requests the selection of the SPARQL query variation that best represents the original NL question is as follows:

```

System:
Consider the following RDF graph written in
Turtle syntax:
{KG_Subgraph}
Given the question: {question}
Select the number of the option that better
represents a SPARQL query for the given question
:
``json
0:{
  "query":"``sparql {SPARQL 0}``,
  "result": {Result Dataset 0}
},
...``
Use the following criteria to evaluate the
options:
-{Restrictions List}
Return only the number of the selected option!

```

Algorithm 5: SelectTriples

```

Input: endpoint, visited_nodes, selected_resources,
n_hops
Output: triples
1 triples ← string();
2 for selected_resource in selected_resources do
3   if not visited_nodes.contains(selected_resource)
4     then
5     visited_nodes.add(selected_resource);
6     triples ← triples +
7     endpoint.get_triples(selected_resource,
8     edge_hops, visited_nodes);
9   end
10 end
11 triples ← SelectTriples(endpoint, visited_nodes,
12 selected_resources, visited_nodes,n_hops= n_hops-1);
13 return triples;

```

The prompt template again starts with the KG subgraph obtained from the fragments of terms relevant to the NL question. It continues with the user’s original NL question, along with the list of SPARQL query variations and the set of results of their execution on the KG. The prompt template ends with a list of restrictions as before.

Finally, below is the prompt template (line 23 from Algorithm 3) generated for the request to generate the final response in natural language for the user. In this prompt the user’s original question is passed. Furthermore, the chosen SPARQL query is passed, accompanied by the set of results from its execution on the KG.

```

System:
Use the SPARQL query and its result set as JSON
object to write a answer to the user question.
DO NOT explain neither cite the SPARQL query and
JSON in yours response.
User question: {question};
SPARQL query:
``sparql
{SPARQL choosed}
``;
JSON result set:
``json
{Result set}
``;

```

V. RESULTS AND DISCUSSION

This section discusses the results of the experiments carried out in the paper, summarized in Table II.

Preliminary Experiments. Consider first the preliminary experiments reported in Sections III-B, III-C and III-D, which correspond to Columns (1) – (4). They indicate that the direct approach, that is, asking ChatGPT to generate answers directly from the NL question and the KG, passed as input, was less accurate than the text-to-SPARQL approach. Indeed, the direct approach (corresponding to Column (1)) correctly answered 10 of the 15 questions asked, achieving an accuracy of 67%,

TABLE II
SUMMARY OF EXPERIMENT RESULTS

Id	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Q ₁	1	1	1	0	1	1	1	1
Q ₂	0	1	1	0	1	1	1	1
Q ₃	1	1	1	0	1	1	1	0
Q ₄	1	1	1	0	1	1	1	1
Q ₅	1	1	1	0	1	1	1	1
Q ₆	1	1	1	0	1	1	1	1
Q ₇	1	1	0	0	1	1	1	0
Q ₈	0	1	0	0	1	1	1	1
Q ₉	1	1	1	0	1	1	0	0
Q ₁₀	0	1	1	0	0	0	0	1
Q ₁₁	0	1	1	0	0	1	1	0
Q ₁₂	1	0	0	0	1	1	0	0
Q ₁₃	1	1	0	0	1	1	1	1
Q ₁₄	0	0	1	0	1	1	1	1
Q ₁₅	1	1	1	1	1	1	1	1
#Correct	10	13	11	1	13	14	12	10
Accuracy	67%	87%	73%	6,7%	87%	93%	80%	67%

Note: (1) NL; (2) T-Box + A-Box; (3) T-Box; (4) A-Box;
(5) Auto-KGQAGPT (GTP-3.5 Turbo); (6) Auto-KGQAGPT (GTP-4);
(7) Auto-KGQAGPT (GTP-4 Turbo); (8) Langchain RAG (GTP-3.5 Turbo).

while the best text-to-SPARQL approach (corresponding to Column (2)) correctly answered 13 of the 15 questions asked, achieving an accuracy of 87%.

A careful observation of the answers of the direct approach⁶ (again corresponding to Column (1)) indicates that, although in many cases the LLM followed a correct reasoning to answer the question, the conclusion reached was incorrect. This becomes evident mainly in NL queries that involve mathematical calculations or comparisons.

Now, a careful observation of the answers of the text-to-SPARQL approach (corresponding to Columns (2), (3) and (4)) stresses the relevance of passing elements from both the T-Box and the A-Box.

When both the T-Box and the A-Box were passed (corresponding to Column (2)), ChatGPT with GTP-3.5 correctly processed 13 NL queries out of 15, achieving an accuracy of 87%. An analysis of the answers⁷ indicates that it generated SPARQL queries that included properties not defined in the T-Box by using fragments of the A-Box.

When only the T-Box was passed (corresponding to Column (3)), ChatGPT correctly processed 11 NL queries out of 15, achieving a lower accuracy of 73%. An analysis of the answers⁸ indicates that the main source of errors was the fact that the T-Box does not explicitly define all the properties used or conventions followed by the A-Box. This scenario is close to real-world scenarios, where the T-Box does not exhaustive model the entire domain, either due to the reuse of concepts from other ontologies or due to incomplete knowledge during the T-Box modeling. Therefore, the importance of the A-Box triples for generating SPARQL queries becomes apparent, especially considering that triples may be used to write the SPARQL queries simply by replacing elements of a triple by variables in the query to be generated.

When only the A-Box was passed (corresponding to Column (4)), ChatGPT correctly processed only 1 NL question out of 15. An analysis of the answers⁹ indicates that the main source of errors was the incorrect use of property URIs, using prefixes incorrectly, although the structure of the SPARQL query was correct.

From a more general perspective, the preliminary experiments also suggest that the text-to-SPARQL approach tends to be better suitable for real-world scenarios with massive KGs, given that: (1) The approach prevents hallucinations during the reasoning process; (2) It allows the evaluation of large sets of instances; (3) It allows outputting large results; (4) It allows omitting sections of the KG to the LLM.

Experiments with Auto-KGQAGP. The results obtained with Auto-KGQAGP, introduced in Section IV, running with GTP-3.5 Turbo, GTP-4, and GPT-4 Turbo, correspond to Columns (5), (6) and (7), respectively. The experiments with these

models can be found at the link¹⁰, along with the results of experiments using Langchain’s RAG, discussed later on.

Auto-KGQAGPT with GPT-3.5 Turbo generated SPARQL queries that correctly translated 13 NL questions out of the 15 tested, the same accuracy as the best manual experiments. Auto-KGQAGPT with GPT-4 generated SPARQL queries that correctly translated 14 NL questions out of the 15 tested, standing out as the best model, with an accuracy of 93% in the dataset. Auto-KGQAGPT with GPT-4 Turbo generated SPARQL queries that correctly translated 12 NL questions out of the 15 tested, achieving a worse result than GPT-3.5 Turbo. An analysis of the NL responses generated using GPT-4 Turbo indicates that the responses were much more complex and detailed than those obtained with GPT-3.5 Turbo.

In general, a close evaluation of the results indicates that the main sources of errors were the incorrect use of property namespaces and the absence of OPTIONAL clauses in optional properties. Furthermore, Q_{10} “Who is over 30?” was incorrectly translated in all models. This is because Q_{10} does not have any explicit reference to elements of the KG, resulting in prompting the LLM with an empty KG fragment. However, when evaluating Q_{10} in manual scenarios, which considered the context of the dialogue, ChatGPT was able to infer that Q_{10} referred to people’s ages. Q_{10} was written with an anaphora on purpose, which requires the history of the conversation to be interpreted. When resubmitting the question rewritten as “Who are the people over the age of 30?”, all three models were able to translate it correctly. GPT-4 generated the best response, in terms of simplicity and completeness.

Experiments with Langchain. New experiments were executed to compare the performance of Auto-KGQAGPT with Langchain’s RAG¹¹ (corresponding to Column (8)) and with Langchain GraphSparqlQChain¹².

Langchain’s RAG was used to index the KG triples, inserted as documents; at run time, the top 90 documents (triples) relevant to the input NL question were retrieved. These documents were then converted to turtle format and passed as a KG fragment to the LLM, which was asked to generate a single SPARQL query translation. The number of triples retrieved was set to 90 since it obtained the best results, without exceeding the number of average tokens used in the Auto-KGQAGPT approach. This approach used the GPT-3.5 Turbo model and generated 10 correct answers.

The results of the experiments with Langchain GraphSparqlQChain¹³ were very poor and not included in Table II.

Reduction in the Number of Tokens Passed. The toy KG of Section III-A, without inferences, requires 2,977 input tokens to pass to ChatGPT, whereas Auto-KGQAGPT required 2,034.3 input tokens on average, which represents a reduction of 31.66% in the number of input tokens.

⁶<https://bit.ly/3RIvqff>

⁷<https://bit.ly/3TrWgtx>

⁸<https://bit.ly/41shQQK>

⁹<https://bit.ly/3v2J58j>

¹⁰<https://bit.ly/3twszwI>

¹¹https://python.langchain.com/docs/expression_language/cookbook/retrieval

¹²https://python.langchain.com/docs/use_cases/graph/graph_sparql_qa

¹³https://python.langchain.com/docs/use_cases/graph/graph_sparql_qa

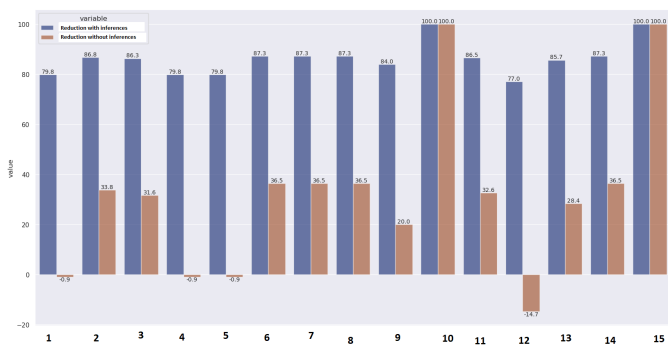


Fig. 2. Percentage reduction in the number of tokens passed per question compared to KG with and without inferences.

Consider now using the reasoner OWL 2 QL¹⁴ to infer triples. The toy KG plus the inferred triples now requires 14,879 input tokens, which makes it impossible to pass to ChatGPT, stressing the importance of the KG fragmentation approach that Auto-KGQAGPT adopts. Indeed, Auto-KGQAGPT generated an 86.32% decrease in the number of tokens, when considering the KG plus the inferred triples.

The Langchain’s RAG approach used an average of 1,531.6 input tokens, indicating that the number of tokens passed by Auto-KGQAGPT can still be decreased, when considering selection at the triple level.

Figure 2 shows the reduction in the number of input tokens for each NL question, when considering the toy KG with inferences (blue - left column) and the toy KG without inferences (orange - right column). The figure shows that an NL question had a negative reduction, that is, an increase in the number of tokens passed. This is due to passing inferred triples to the LLM with the fragments of the KG. These inferred triples act as non-explicit knowledge contained in the KG.

VI. CONCLUSIONS

This paper presented preliminary experiments to evaluate the ability of ChatGPT (with GPT-3.5) to answer NL questions over a KG. The experiments suggest that the text-to-SPARQL approach, that is, generating a SPARQL translation of the NL question, passing both the T-box and the A-box, achieved better results among the scenarios evaluated.

Based on these experiments, the paper introduced Auto-KGQAGPT, an autonomous domain-independent framework based on LLMs for text-to-SPARQL. The main goal of Auto-KGQA was the selection of smaller KG fragments thereby reducing the number of tokens passed as input to the LLM. Experiments with Auto-KGQAGPT suggest that the framework achieved this goal, without sacrificing performance.

Thus, the preliminary experiments reported in this paper suggest that LLMs are a promising technology for constructing KGQA systems, being able to answer NL questions for specific KGs, without requiring large volumes of training data.

Future work includes the evaluation of new matching mechanisms for KG elements at the input, the selection of KG fragments at the triple level, the iterative correction of the SPARQL query generated, an evaluation of fine-tuning techniques to improve results, and testing Auto-KGQAGPT on large KGQA benchmarks.

ACKNOWLEDGMENT

This work was partly funded by FAPERJ under grants E-26/200.834/2021, by CAPES under grant 88881.134081/2016-01 and 88882.164913/2010-01, and by CNPq under grant 305.587/2021-8.

REFERENCES

- [1] L. Hirschman and R. Gaizauskas, “Natural language question answering: the view from here,” *natural language engineering*, vol. 7, no. 4, pp. 275–300, 2001.
- [2] M. Yani and A. A. Krisnadhi, “Challenges, techniques, and trends of simple knowledge graph question answering: a survey,” *Information*, vol. 12, no. 7, p. 271, 2021.
- [3] D. Diefenbach, V. Lopez, K. Singh, and P. Maret, “Core techniques of question answering systems over knowledge bases: a survey,” *Knowledge and Information systems*, vol. 55, no. 3, pp. 529–569, 2018.
- [4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, “Language models are few-shot learners,” *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.
- [5] Y. Tan, D. Min, Y. Li, W. Li, N. Hu, Y. Chen, and G. Qi, “Can chatgpt replace traditional kbqa models? an in-depth analysis of the question answering performance of the gpt llm family,” in *International Semantic Web Conference*. Springer, 2023, pp. 348–367.
- [6] M. R. A. H. Rony, U. Kumar, R. Teucher, L. Kovriguina, and J. Lehmann, “Sgpt: a generative approach for sparql query generation from natural language questions,” *IEEE Access*, vol. 10, pp. 70712–70723, 2022.
- [7] Z. Qi, Y. Yu, M. Tu, J. Tan, and Y. Huang, “Foodgpt: A large language model in food testing domain with incremental pre-training and knowledge graph prompt,” *arXiv preprint arXiv:2308.10173*, 2023.
- [8] C. Feng, X. Zhang, and Z. Fei, “Knowledge solver: Teaching llms to search for domain knowledge from knowledge graphs,” *arXiv preprint arXiv:2309.03118*, 2023.
- [9] L. Kovriguina, R. Teucher, D. Radyush, and D. Mouromtsev, “SPARQLGEN: one-shot prompt-based approach for SPARQL query generation,” in *Proc. of the Posters and Demo Track of the 19th International Conference on Semantic Systems co-located with 19th International Conference on Semantic Systems (SEMANTiCS 2023), Leipzig, Germany, September 20 to 22, 2023*, ser. CEUR Workshop Proceedings, N. Keshan, S. Neumaier, A. L. Gentile, and S. Vahdati, Eds., vol. 3526. CEUR-WS.org, 2023. [Online]. Available: <https://ceur-ws.org/Vol-3526/paper-08.pdf>
- [10] D. Beckett, T. Berners-Lee, E. Prud’hommeaux, and G. Carothers, “Rdf 1.1 turtle,” *World Wide Web Consortium*, pp. 18–31, 2014.
- [11] Elastic, “Elasticsearch,” <https://www.elastic.co/elasticsearch/>, Acesso em 23 de novembro de 2023.
- [12] J. C. A. de Figueiredo, “Tutorial de manipulação de strings,” <http://www.dsc.ufcg.edu.br/~abrantes/CursosAnteriores/ATAL031/String/String.html>, 2007, acesso em 23 de novembro de 2023.

¹⁴https://www.w3.org/TR/owl2-profiles/#OWL_2_QL_2