SemanticLang: Infinite-length document information extraction for traceable question answering

Yuan Ma

High School Affiliated to Renmin University of China

Email: semanticlang@kip.gay

Abstract

Document Question-answering (QA) is an important task in the field of neural language processing. It requires the computer to extract information from text and answer the user's questions accurately based on that information. Transformer-based large language models (LLMs) have proved to perform well in this task, as they are powerful in text understanding and have flexible logic deduction skills, which are required for cohesive question-answering. However, hardware memory and computation restrictions often limit the accuracy of the purely transformer-based models' logic deduction process. Explainability of AI is crucial to allow the model's deduction process to be supervised and verified to be correct. Typical neural networks' un-explainability cause large language models to be unaccountable in mission-critical tasks. Therefore, lightweight QA models with user-viewable deduction processes are crucial to creating explainable AI that's accountable. In this project, we aim to create an evidence-first lightweight model that can have its logic deduction process verified by users. We created a hybrid AI system for document knowledge question-answering that answers the user's questions by constructing a knowledge graph that can be explored by the user or/and be used by SemanticLang to answer the user's questions, and a chain of deduction would be tracked that will allow the user to trace the answer back to the evidence in text. The system consists of three Gemma2-2b/Gemma2-9b transformer language models fine-tuned with modified WebNLG and vquanda datasets, they are combined with a modified Resource Description Framework (RDF) triple store to create a lightweight AI system for information extraction and question answering. It combines the language understanding capabilities of language models and the reliability and transparency of RDF. We introduce a novel approach called "Multi context" that improves the explainability and accuracy of the model's understanding process, it also allows identification of specific evidence used for deduction, improves parallelization and unlocks theoretically infinite context length for the model. We evaluated the performance of SemanticLang's information extraction and question answering ability and conclude that it has comparable or higher accuracy

than mainstream lightweight or even online LLMs, additionally SemanticLang provides unique reliable explainability, knowledge exploration and infinite context length characteristics not found in mainstream LLMs.

The source code of the project is available at https://kip.gay/SemanticLang.

Index terms: Explainable AI, Information Extraction, Question Answering, Transformer, Large Language Model, Resource Description Framework

Contents

1. Introduction

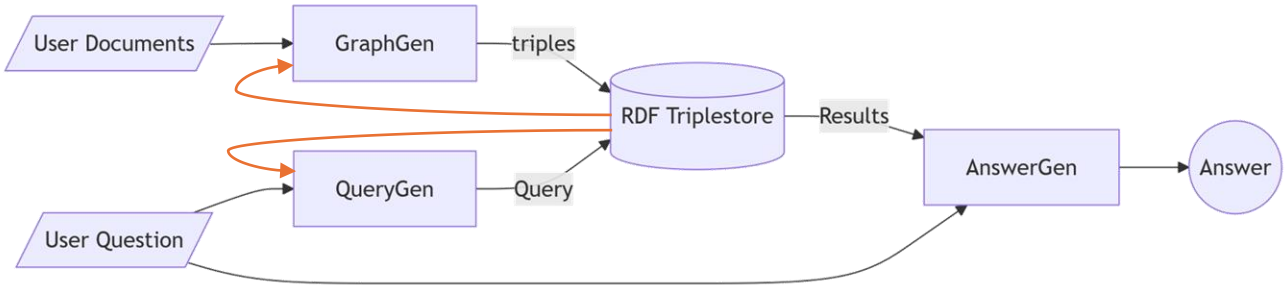

Fig. 1: Overview of the SemanticLang system structure

Question-answering (QA) is a fundamental task in natural language processing, requiring systems to extract relevant information from text and provide accurate, contextually relevant answers to users' questions. One of the common use-case of a LLM is for it to be used as a search engine-like question answering system[1]. While some models rely on internal memory to answer questions, using webpages or documents to gather up-to-date information is better for accuracy and spread of knowledge. This requires models to understand document(s) with long text length.

With the rise of transformer-based large language models (LLMs), substantial improvements have been made on the flexibility and accuracy of QA tasks. These models, such as OpenAI's GPT-4[2] have demonstrated significant abilities on text comprehension and flexible reasoning, making them strong candidates for answering complex queries[2].

However, despite their success, purely transformer-based language models face several limitations, particularly when tasked with handling in-depth logical deductions on long documents.[3] Transformers' main goal is to extract and manipulate semantic meaning in text, therefore extracting logic from text requires considerable levels of abstraction, requiring more layers. [4]The high computational and memory costs of the attention mechanism in transformers also pose challenges, especially when run locally where resources are limited, often resulting in suboptimal performance in terms of both accuracy and efficiency.[5]

Furthermore, the "black box" nature of neural networks, particularly with the scale of LLMs, raises concerns around explainability. The neural nature of LLMs make their deduction process unexplainable to users, humans are unable to understand or find problems in the internal reasoning process of these models[6], [7], which hinders trust

and transparency. Many LLMs are designed to be general purpose models, as their scale increases, internal memory may introduce more unwanted content in the output. The un-explainability of models makes them unaccountable and unverifiable on logic deduction, limiting their future application on mission-critical tasks.[8]

To address these challenges of conventional Transformer-based language models, this paper presents a novel, evidence-first lightweight Q&A model called SemanticLang. It combines a conventional language model with knowledge graphs to improve the model's explainability, performance and context length (Fig. 1).

2. Related Work

Large Language Model with Prompt Engineering. Chain-of-Thought prompting[9], Least-to-Most prompting[10] are prompt engineering techniques that encourage LLMs to process a single problem in multiple sub-steps to produce more logical answers. These in-context learning approaches reduce hallucinations and improve the LLM's output accuracy, they do not fix the underlying problem of hallucinations and logic limitations coming from the Transformer architecture.[11] Most Large language models also have limited context length, which restricts the number of documents the model can understand for a question. Large language models are also not easily explainable. LLM's complexity makes it hard to trace the logic deduction process on the low level, and steps generated by the model through prompt engineering may be incorrect. This paper's model offloads the logic deduction process to the Resource Description Framework Graph, only requiring the language model to understand semantic information from text. Our system's hybrid and multi-stage process allows the deduction process to be clearly visualized.

Natural Language to Predicate Logic. There has been research on offloading pure logic deduction in Transformers to Logic Solvers, such as using language models to translate text in natural language to first-order logic. Logic solvers are then used to process the logic statements and produce a result. [12], [13]These strategies can effectively improve the logic deduction capabilities of LLMs and perform well in solving logic problems, but they are not well suited to the use-case of information extraction and question answering.

Resource Description Framework. Resource Description Framework[14] is a part of the Semantic Web. Semantic Web, also known as Web 3.0 (Not to be confused with Web3) was proposed in 1999 as an extension to the World Wide Web. It included several technologies that introduced a standardized data structure that "allows data to be shared and reused across application, enterprise, and community boundaries[.]"[14] One of the key components of the Semantic Web is the Resource Description Framework, which

defines the RDF Triple (or triple)[15]: A sequence of three keywords in the order of "subject–predicate–object" that describes a directional relationship between things. Multiple triples that share the same keywords can create a network of relationships, the network can then be stored in a relational database where it can be queried against using an RDF query language such as SPARQL[16], allowing the system to reply to the query using the stored knowledge. RDF is used in our project as the main system that organizes knowledge and performs deduction.

3. Our Method

SemanticLang consists of three modules: **GraphGen, QueryGen** and **AnswerGen**, with GraphGen being the focus of the project. SemanticLang is a Q&A system that can understand documents written in natural language (NL) and answer user's questions based on the information in the documents. SemanticLang's design revolves around RDF's ability to represent and connect knowledge in a structural manner, and that RDF has a query (pattern matching) language called SPARQL that can successfully represent and answer many types of questions.[15] SemanticLang process a question as follows (Fig. 2):

1. **GraphGen:** It combines the custom fine-tuned Gemma2-2b[17]/Gemma2-9b language model with a novel technique we call "Multi Context" to understand the text documents and extracts the knowledge in the documents as RDF-like triples where things' relations and attributes are expressed formally. The network formed by the triples stored in an RDF triple store. Each triple is assigned an ID that indicates the document and sentence from which the triple was generated. The network can also be visualized as a graph to improve the transparency of the understanding process.
2. **QueryGen:** It understands the User's question about the contents and converts the question into a standard SPARQL query. Our implementation includes systems that ensure the model generates a valid SPARQL query with high relevancy to the generated network.
3. **RDF Triple store:** It queries the network with the SPARQL string generated by QueryGen and returns the result.
4. **AnswerGen:** uses the context of the question and the result from the SPARQL query to answer the user's question contextually.

These modules, when connected create a complete seq2seq QA model that takes in documents and a question, then returns the answer. Visualization of the generated RDF graph allows the user to explore knowledge originally as text in the documents in an interactive graph and to verify the model's generation. Additionally, using the triples' evidence ID and the SPARQL query, the system can backtrack from the answer to the

specific chunks of text used for generation of the triples used in the query, which can be presented to the user as evidence of the model's answer. The GraphGen module can also be run independently to create graphs for knowledge visualization or create RDF-like triple store for other processing tasks.

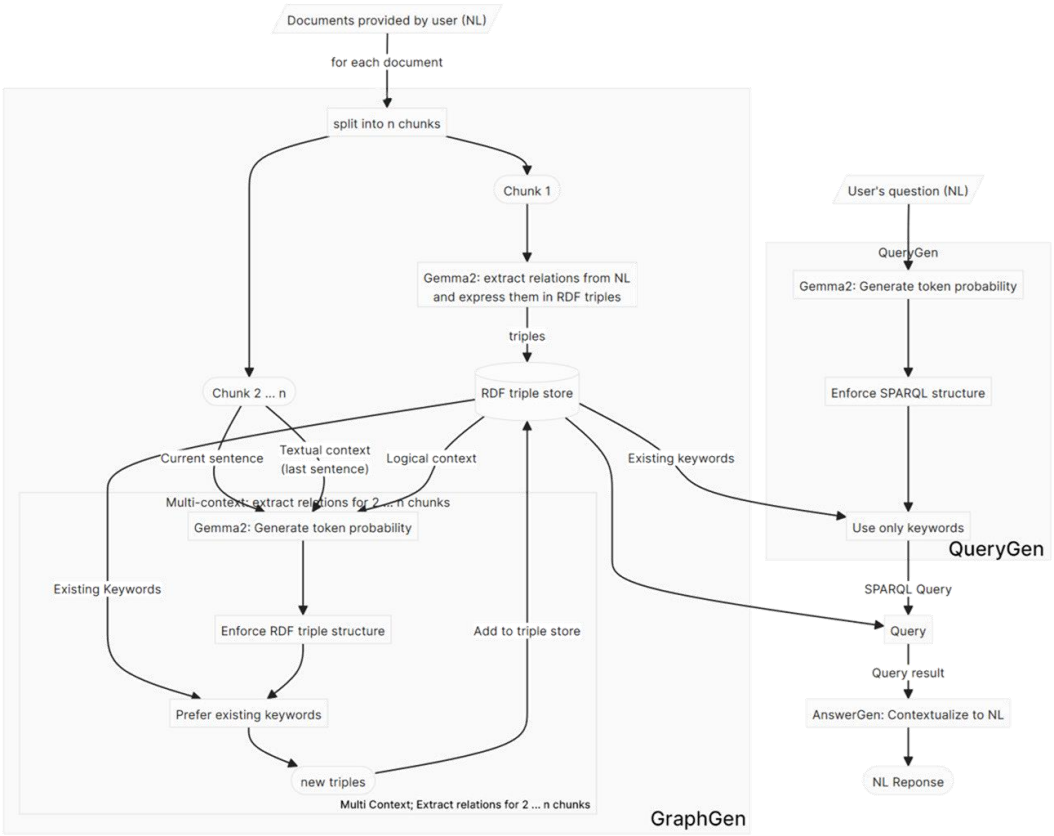

Fig. 2: Graph representing the structure of processing done in SemanticLang

3.1. GraphGen & Multi Context

Understanding the information represented in documents is crucial for document QA models. Because of the quadratic complexity of the attention mechanism as context length increase[4], traditional transformer-based language models' context length is limited by hardware restrictions, which limits the number of tokens, or the length of documents the model can process. This presents a challenge for models to understand long documents or multiple documents.

With long documents, it's hard for humans to manually verify a model's output accuracy through the answer alone. LLMs may produce invalid sources and reasoning steps when asked. For example, when ChatGPT with gpt-4o[18] was presented the lead section of Wikipedia pages for *"World Wide Web"*[19], *"Semantic Web"*[20] and

"Resource Description Framework"[21] and asked *"List all components of the semantic web. List your evidence."*, ChatGPT returned evidence sentences such as *"Technologies such as Resource Description Framework (RDF) and Web Ontology Language (OWL) are used to formally represent metadata.*[1]*"*, which do not appear in the provided text.

We have developed GraphGen with Multi Context to address the problems mentioned above. GraphGen splits a document into *n* chunks, where each chunk is processed by the language model independently. Each chunk is around 400 characters long, this addresses the problem of limited context length, but it removes context that came from other parts of the text which could decrease the model's accuracy. Multi Context addresses the context problem by simultaneously using three techniques to add context to the sentence:

Textual Context: We theorize that the most relevant textual context of a sentence comes from sentences immediately before it. Fig. 3 shows one such example.

"The World Wide Web (WWW or simply the Web) is an information system that enables content sharing over the Internet through user-friendly ways meant to appeal to users beyond IT specialists and hobbyists. It allows documents and other web resources to be accessed over the Internet according to specific rules of the Hypertext Transfer Protocol (HTTP)."
The "It" in red refers to the World Wide Web. This important information would be lost if the yellow sentence was viewed alone.

Fig. 3: An example of textual context coming from the neighboring sentence.

During chunk generation, the last sentence of the previous chunk is added to every chunk of text. It creates an overlap of processed text which can provide important textual context (Fig. 4).

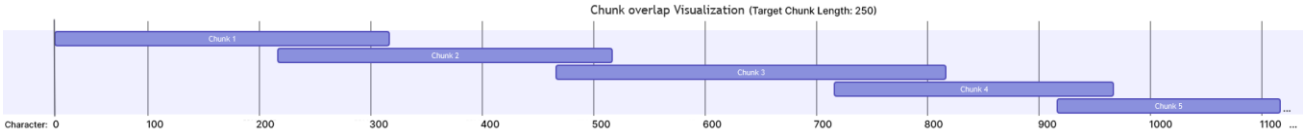

Fig. 4: Demo visualization of chunk overlap when target chunk length is set to 250.

Semantic Context: Since previous sentences have been converted into RDF triples, these triples in the triple store can provide semantic (logical) context in a modular and space efficient way. When a chunk is processed by the model, a random selection of 10 triples generated from the previous chunks of the document is also included as part of the prompt. This enables the model to gain semantic context of the entirety of the document.

[1] The correct text is *"technologies such as Resource Description Framework (RDF)[2] and Web Ontology Language (OWL)[3] are used."*. Original text from Semantic Web article on Wikipedia. [20]

Keyword Reward: RDF triples mostly use Universal Resource Identifiers (URIs)[15]. Because the triples made by GraphGen are generated from NL documents, we use plain text instead of URIs for the naming of subject, object and predicates. We refer to the subject and objects as *nodes*, and the names of subject, object and predicates in triples as *keywords*. We found that our original model may generate triples with the same semantic meaning but with keywords with different wording. RDF triple store cannot recognize the similarity between the two triples, instead it generates two unique nodes. This raises a problem as it drastically reduces connectivity in the graph, reducing the accuracy of queries. The problem worsens when multiple documents are parsed. Each document may use different keywords to represent the same node, making querying across multiple documents difficult. We developed Keyword Reward to decrease the chance of this happening: A custom Logit Processor guides the model to use existing keywords instead of creating new ones.

Multi Context provides context through multiple info types in the prompt and enforces context during the token generation process. Because of the relatively short context length achieved by Multi Context and scalability of RDF triple stores, GraphGen's theoretically able to handle documents of any length. GraphGen is theoretically parallelizable, the RDF triple store can be updated in real-time, and the most recent semantic context will be retrieved and used when processing a chunk, allowing multiple chunks to be processed at the same time.

3.1.1. Implementation

Fine-tuning

GraphGen is developed using unsloth's optimized Gemma2-2b model[22], a 2 billion parameter pre-trained transformer-based language model. The relatively small size of the model allows GraphGen to be run locally on PCs. We used a modified version of the WebNLG[23] dataset to fine-tune the Gemma2-2b model for the task of RDF triples generation. Additionally, a 9 billion parameter variant of the model is created by fine-tuning the Gemma2-9b model.

```
<entry category="Airport" eid="Id1" shape="(X (X (X) (X (X))))" shape_type="mixed" size="4">
  <originaltripleset>
    <otriple>Aarhus_Airport | location | Tirstrup</otriple>
    <otriple>Tirstrup | country | Denmark</otriple>
    <otriple>Denmark | capital | Copenhagen</otriple>
    <otriple>Tirstrup | subdivisionName | Central_Denmark_Region</otriple>
  </originaltripleset>
  <modifiedtripleset>
    <mtriple>Aarhus_Airport | location | Tirstrup</mtriple>
    <mtriple>Tirstrup | country | Denmark</mtriple>
    <mtriple>Denmark | capital | Copenhagen</mtriple>
    <mtriple>Tirstrup | isPartOf | Central_Denmark_Region</mtriple>
  </modifiedtripleset>
  <lex comment="good" lid="Id1">Aarhus airport is located in Tirstrup, part of the Central Region of Denmark which has the capital city of
```

```
Copenhagen.</lex>
  <lex comment="good" lid="Id2">Copenhagen is the capital of Denmark where Aarhus airport is located in Tirstrup which is part of the Central
Denmark region.</lex>
  <lex comment="good" lid="Id3">Aarhus Airport is located in Tirstrup, part of the Central Denmark region. The capital of the country is
Copenhagen.</lex>
</entry>
```

Fig. 5: One pair in the WebNLG dataset

The WebNLG dataset consists of datasets of pairs of 1 to 7 triples and each pair contains multiple human generated text annotations (Fig. 5). We modified the dataset to add special tokens <T>, <R> and <S> to indicate the subject, predicate and object parts respectively *(The actual tokens used is <unused0>, <unused1> and <unused2>. For readability these tokens are represented as <T> <R> and <S> in both the code and this paper)*. Triples paired with a text annotation are randomly selected to become semantic context.

Example of a triple in dataset:
`<T>11th_Mississippi_Infantry_Monument<R>category<S>Contributing_property`

Semantic Context	Triples
<T>11th_Mississippi_Infantry_Monument <R>established <S>2000 <T>11th_Mississippi_Infantry_Monument <R>location <S>Adams_County,_Pennsylvania <T>11th_Mississippi_Infantry_Monument <R>municipality <S>Gettysburg,_Pennsylvania	<T>11th_Mississippi_Infantry_Monument <R>category <S>Contributing_property <T>Adams_County,_Pennsylvania <R>hasToItsNorth <S>Cumberland_County,_Pennsylvania <T>11th_Mississippi_Infantry_Monument <R>country <S>"United States"

NL	The 11th Mississippi Infantry Monument, built in 2000, is placed in the municipality of Gettysburg in Pennsylvania which is in Adams County, USA. The 11th Mississippi Infantry Monument is classified as a Contributing Property. Cumberland county, Pennsylvania is to the north of Adams County.

Fig. 6: One pair in the modified dataset

Using this modified dataset (Fig. 6), we generate prompts that combine the semantic context and NL. The prompt and triples are used to fine-tune the Gemma2-2b model. The model was fine-tuned with a total batch size of 8 and fine-tuned for 1000 steps. The 9b model is fine-tuned with a batch size of 8 for 1 epoch (12142 steps).

Inference: Textual Context & Semantic Context

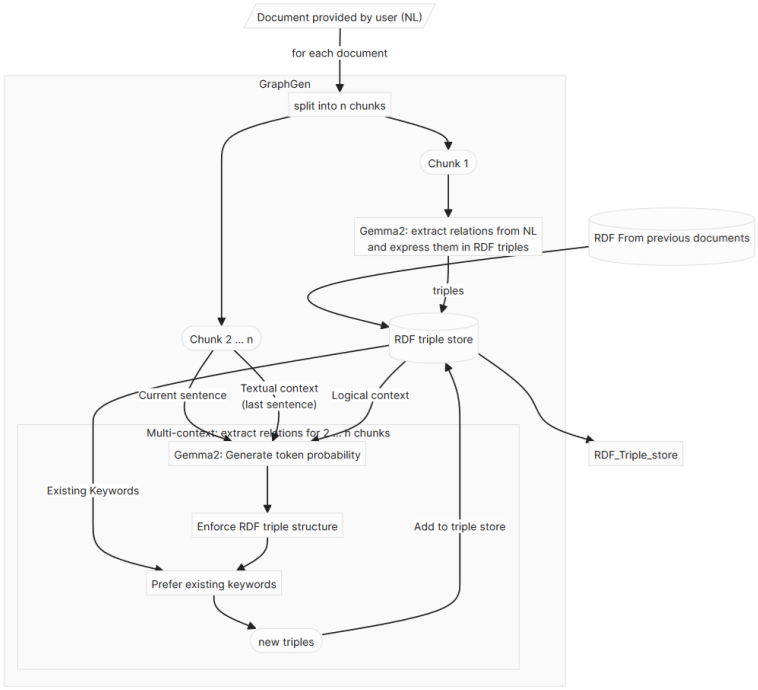

Fig. 7: Architecture of GraphGen's inference process

Inference in GraphGen consists of multiple steps (Fig. 7). Natural Language Toolkit[24] is used to separate a document into a list of sentences. Then these sentences are merged based on the length. For each chunk, the last sentence of the previous chunk is appended to the front of the current chunk to provide textual context. The length of the accounting chunk shall be under the chunk length limit, except if the chunk repeats the last when following this length limit (Fig. 8). By appending the last sentence into the chunk, textual context is achieved.

```
sentences = pkt_tokenizer.tokenize(input_text) # Document is split by
sentences using the nltk punkt tokenizer

merged_sentences = []

for i in range(len(sentences)):
    sentence = sentences[i]
    if len(sentence) <= 0: # Skip empty strings
            continue
        if i >= 1 and len(merged_sentences[-1]) + len(sentence) <=
max_chunk_length:
            # Append current string to the last chunk if it will be below
the length limit
            merged_sentences[-1] += " " + sentence
        else:
            # Create new chunk
            if i >= 1:
                merged_sentences.append(sentences[i - 1]) # Add last
sentence for textual context if applicable
            merged_sentences.append(sentence)
merged_sentences
```

Fig. 8: Pseudo-code for chunk generation algorithm

Each chunk is processed by the fine-tuned Gemma2 model combined with Multi Context. Textual context is achieved during chunk generation, semantic context is added during prompt generation.

Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.
Instruction:
Extract the most confident information in the sentence below as much as possible, and express the relationships in RDF Triples that complement the existing RDF triples. Do not use information from common sense.
Existing RDF triples:
{Semantic Context in the form of triples, or “None” if none exists}
Input:
{Document chunk in natural language. Unnecessary whitespaces are removed.}
Response:
<T>

Fig. 9: Prompt for training and inference in GraphGen.

Semantic Context is presented to the model through the “Existing RDF triples” section of the prompt as seen in Fig. 9. Semantic Context is limited to the current document. When previous chunks are processed, the generated RDF triples are added to the RDF triple store. During prompt generation, 15 triples will be randomly selected from the document’s RDF triple store and added into the prompt (Fig. 10). We intend that this provides the language model the semantic context of the entire document, including key object, subjects and topics in a compressed manner, and that selecting triples randomly can decrease the impact of incorrect triples on the entire generation.

Instruction:

Extract the most confident information in the sentence below as much as possible, and express the relationships in RDF Triples that complement the existing RDF triples. Do not use information from common sense.

Existing RDF triples:

<T>Quiz_bowl<R>location<S>"Nationwide" <T>Quiz_bowl<R>region<S>"Nationwide" <T>Academic_Bowl<R>region<S>"Nationwide" <T>Academic_Bowl<R>country<S>United_States <T>Academic_Bowl<R>sport<S>Quiz_bowl <T>Quiz_bowl<R>gameplay<S>"Buzzer" <T>Academic_Bowl<R>alternativeName<S>"Scholars' Bowl" <T>Quiz_bowl<R>player<S>"team of four" <T>Quiz_bowl<R>alternativeName<S>"Scholars' Bowl", "Academic Bowl", "Academic Team", "Academic Challenge", "Scholastic Bowl", "Primary School Quiz Bowl", "Middle School Quiz Bowl", "High School Quiz Bowl", "University Quiz Bowl"

Input:

A moderator reads questions to the players, who try to score points for their team by buzzing first and responding with the correct answer.

Response:

<T>

Fig. 10: A complete prompt with Semantic Context and Textual Context[2]

RDF Structure Enforcement & Keyword Reward

During the project, we found that the model sometimes generates triples with invalid structures. These invalid structures include incorrect <T> <R> <S> order and placement, Incoherent output and repetition (Fig. 11).

<bos>Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request.

Instruction:

Extract the most confident information in the sentence below as much as possible, and express the relationships in RDF Triples that complement the existing RDF triples. Do not use information from common sense.

Input:

Logits can range from (-∞,+∞) , where a higher value means that the token's more probable to be the next in the sequence.

Response:

<T>Logits<R>range<T>Logits<R>associatedNumber<S>1<T><R><S>1<S><eos>

Fig. 11: An example of incorrect output from an early version of our model

Another problem we found was that our early models may generate nodes with the same semantic meaning but with different keywords (Fig. 11). This is problematic as the RDF is designed to work with URIs and will not recognize two nodes as the same if they

[2] Input sentence from Wikipedia. [25]

have different keywords. Furthermore, similar keywords may have different semantic meaning, e.g. *Complement* vs *Compliment*, so a fuzzy matching algorithm would not be appropriate for merging similar keywords.

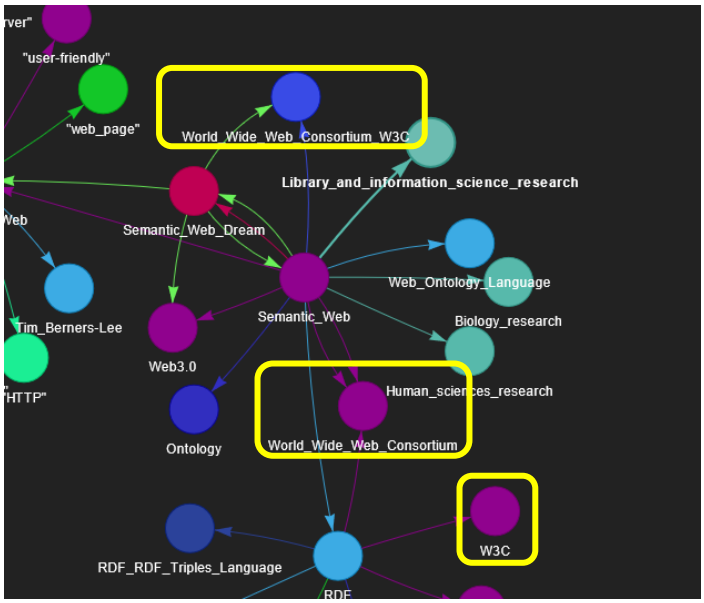

Fig. 11: An example of a poorly generated RDF graph from an early version of our model. Notice the nodes boxed with a yellow outline all represent the World Wide Web Consortium.

Transformer language models generate text by predicting the most probable token following the input sequence. Specifically, in the final layer, a list of logits for all tokens are generated. Logits can range from $(-\infty, +\infty)$, where a higher value means that the token's more probable to be the next in the sequence. The logits usually get normalized by an activation function such as *softmax* into confidence scores which is used for the final token selection.[4] We developed two custom logit processor for GraphGen that modify the model's generated logits before normalization to address the two problems.

TRSLogits. TRSLogits enforces the correct triple structure by disabling invalid special tokens. This allows our model to adapt to the required triple structure and produce valid outputs. Special tokens are used by our RDF parser to deserialize strings into internal triples structures, they're also exploited by the paper's custom logit processors to improve the model's generation quality (Fig. 12).

Special Tokens				
Token	<T> (<unused0>)	<R> (<unused1>)	<S> (<unused2>)	<eos>
Meaning	Start of new triple, start of subject	Start of predicate	Start of object	End of string. Ends generation

Fig. 12: The special tokens used in GraphGen and their meaning

During the model's generation process, after the final layer finishes processing and produces the list of logits, the logits, and the token sequence used for generation is passed to TRSLogitss. TRSLogits finds the special token with the largest index in the sequence and prevents corresponding tokens from being generated based on the type of the special token. Note that tokens and special tokens are presented as their detokenized text counterpart in this paper. Gemma2-2b uses sentencepiece[26] for tokenization which includes byte-pair-encoding. Our special tokens are all 1 token long.

TRSLogits checks the closest special token from the generation position, then disables certain special tokens based on the closest special token (Fig. 13). For example, the model cannot generate TRS tags out of order, so <S> can't be selected when <T> has just been generated, and special tokens cannot be adjacent (it cannot generate triples with no content).

```
func TRSLogits.Process(original_logits, sequence):
        t_largest_index   = GetLargestIndex(t_token)    # <T>
        r_largest_index   = GetLargestIndex(r_token)    # <R>
        s_largest_index   = GetLargestIndex(s_token)    # <S>

        eos_largest_index = GetLargestIndex(eos_token) # <eos>

        max_index = max(t_largest_index, r_largest_index,
                        s_largest_index, eos_largest_index)
        # Find which special token is the closest to the token we're generating

        if max_index == len(sequence) - 1:
                # Just generated a special token, no conseq. special tokens allowed
                disabled_tokens = [t_token, r_token, s_token, eos_token] # ALL NO
        if max_index == t_largest_index: # <T>
                disabled_tokens = [t_token,          s_token, eos_token] # <R> OK
        if max_index == r_largest_index: # <R>
                disabled_tokens = [t_token, r_token,          eos_token] # <S> OK
        if max_index == s_largest_index:
                # Either stop generating or create a new triple
                disabled_tokens = [         r_token, s_token           ]

        return LogitsAfterDisablingTokens(original_logits, disabled_tokens)
```

Fig. 13: Pseudo-code of TRSLogits

We disable tokens by setting their respective logits to $-\infty$, preventing them from being chosen as the predicted token. TRSLogits, through enforcing the subject-predicate-object sequence and generation of full triples before stopping inference, forces the model to generate full, valid triples.

KeywordReward: KeywordReward, or *PreferKeywordsLogit*, decreases the chance of multiple keywords for the same node being generated by recommending existing keywords to the model through modifying the logits. From the RDF triple store, two sets of keywords are extracted:

$$S_{nodes} = \{s_{node_1}, s_{node_2}, \dots, s_{node_n}\}$$

for node keywords (subject and object in triples) and

$$S_{predicates} = \{s_{predicate_1}, s_{predicate_2}, \dots, s_{predicate_n}\}$$

for predicates in the triples. For each set of keywords, the keywords are tokenized,

$$Tokenize(S) \rightarrow \{k_1, k_2, \dots, k_i\}$$

$$K_{tokenized} = \{Tokenize(s_1), Tokenize(s_2), \dots, Tokenize(s_n)\}$$

and an out tree of tokens is generated with root <T>, <R> or <S>, pointing towards all possible 1st tokens of keywords (Fig. 13-B). For node A of each level, it points to all possible subsequent tokens that may be generated when the previously generated tokens are the nodes from root to A, in that order.

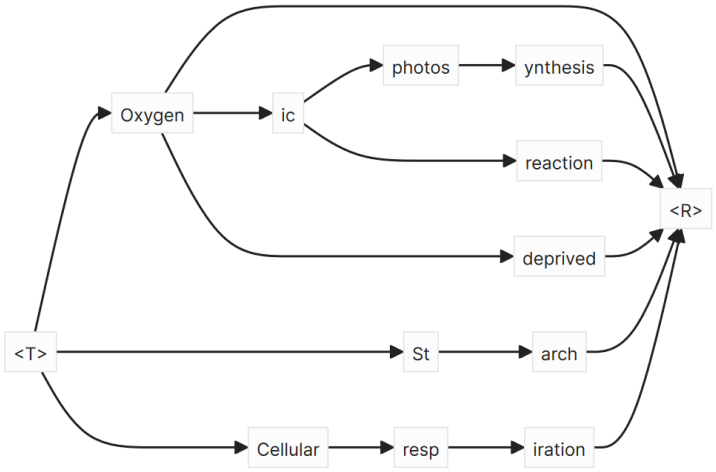

Fig. 13-B: Visualized example of a generated token reward tree

After generating logits, the logits are modified by KeywordReward. KeywordReward uses the same algorithm as TRSLogits to detect the most recent special token, and modifies the logit based on the type of the special token.

Let the previous token sequence starting from special token be $K_{previousTokens} = \{k_1, k_2, \dots, k_n\}$, logits be $P = \{p_1, p_2, \dots, p_n\}$, and the token reward tree be T. We declare

$$Next(T, K_{previousTokens}) \rightarrow K_{ProbableTokens} = \{k_1, k_2, \dots, k_n\}$$

function to traverses the token tree based on already generated tokens. It returns a set of possible immediately subsequent tokens from the tree where it routes from root matches the already generated tokens or returns an empty set if one or more of the already

generated tokens fall outside of the tree.

We then reward these probable existing keyword tokens. Because logits can be either positive or negative, we developed an algorithm that increases the logit proportionally. This promotes high-ranking existing tokens to be more likely to be selected than slightly higher-ranking tokens that have not been selected before.

$$RewardToken(p, \mu_{reward}) = p + (\mu_{reward} - 1) * |p|$$

Where

If the special token is the subject <T> or object <S> token:

$$P' = \{ p_i \mid i \notin Next(T_{node}, K_{previousTokens}) \} \cup \{ RewardToken(p_i, n_{reward}) \mid i \in Next(T_{node}, K_{previousTokens}) \}$$

If the special token is the predicate <R> token:

$$P' = \{ p_i \mid i \notin Next(T_{node}, K_{previousTokens}) \} \cup \{ RewardToken(p_i, n_{reward}) \mid i \in Next(T_{predicate}, K_{previousTokens}) \}$$

P' replaces P as the final logits. This procedure reduces the chance where different keywords representing the same node gets generated, improving the RDF graph's connectivity and the query-ability and generalization ability of the triple store.

These two custom logit processors run in serial, with the TRSLogits' processed logits being fed into KeywordReward to produce the final logits, addresses the problems of the model's tendency to generate triples with incorrect structure and/or unnecessary new keywords.

Deserialization

The triples string generated by QueryGen is parsed into rdflib[27] triples (using URIRef with the keywords as fake URIs). rdflib provides interfaces to create, manage and query RDF databases in python. Simultaneously the triples are stored using an internal format. Here, a sentence ID and document ID is stored in the triples object, allowing the user to check the evidence sentence from which the triple is generated from. Using pyvis[28], we generate an interactive directional graph that visualizes the relations between objects and a reference to the evidence of the relation in text. The color of the arrow represents the sentence from which it is generated from. Fig. 14 shows a

demonstration of a piece of document being converted into triples and visualized.

> Natural language processing (NLP) is an interdisciplinary subfield of computer science and artificial intelligence. It is primarily concerned with providing computers with the ability to process data encoded in natural language and is thus closely related to information retrieval, knowledge representation and computational linguistics, a subfield of linguistics. Typically data is collected in text corpora, using either rule-based, statistical or neural-based approaches in machine learning and deep learning.
>
> […]
>
> Text-to-video
>
> Given a description of a video, generate a video that matches the description.[44][45]
>
> ***(https://en.wikipedia.org/wiki/Natural_language_processing), 26470 characters***

Fig. 14-A: Document provided to GraphGen

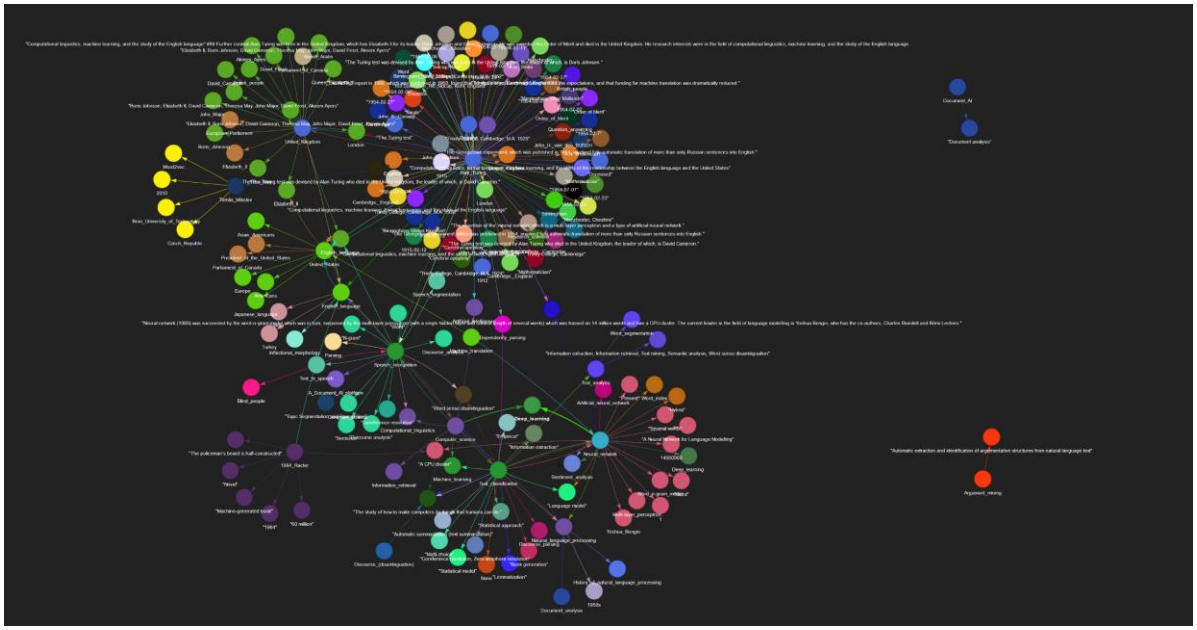

Fig. 14-B: Image of the generated visualized graph. Colors represent the source sentence from which the triple is generated from. The interactive graph allows the user to hover on the arrow to see the predicate.

3.2. QueryGen

To create a complete seq2seq document QA model, we have created two other models to complete GraphGen. RDF triple store can be queried using an RDF query language.

We have chosen the standard RDF query language SPARQL for its flexibility and availability of training datasets. SPARQL has a complicated syntax that's not like typical natural language conversations, and they require specific keywords in the query that are generated by GraphGen, making it hard for users to manually create SPARQL queries. QueryGen converts a user's question in NL into a SPARQL query based on the context of the RDF graph. It is a Gemma2-2b language model finetuned with the vquanda[29] dataset combined with a custom logit processor: SPARQL Enforce & Keyword Enforce.

SPARQL

SPARQL[16] is an RDF query language with a similar syntax to database query languages such as SQL. Users can use SPARQL to ask relatively complex question about the knowledge stored in SPARQL endpoints (in this case, the RDF triple store)

```
SELECT DISTINCT ?uri WHERE
{
?x <residence> <British_Columbia> .
?x <religion> ?uri  .
?x <type> <Politician>
}
```

Fig. 15: An example of a SPARQL query in the vquanda dataset.

The basic syntax of SPARQL is as follows: Whitespaces have no effect in SPARQL, instead brackets and triple separators(".") are used. The first word indicates the type of query performed. SELECT queries extract raw data from the database. CONSTRUCT returns an RDF graph based on the graph template in the query. ASK queries returns if the query has a solution. DESCRIBE returns an RDF graph generated by the RDF database. Variables can be declared in the form of *"?variable_name"*. The WHERE{} adds conditions to the query. In the conditions exists triple-like structures following the subject-predicate-object order. Keywords in brackets represent an existing keyword (Usually URIs, this project modified them into plain strings). Any of the three positions in a triple can be replaced with a variable. Special functions like COUNT() adds additional functionality and conditions to the query.

We use the SPARQL query showcased in Fig. 14 as an example. It requires the RDF triple store to return a list of unique *?uri* where *?x*'s religion is *?uri* and *?x*'s residence is British Columbia and *?x*'s type is Politician.

Fine-tuning

The vquanda dataset consists of pairs of Question in NL-SPARQL-answer in NL. The SPARQL queries in the dataset contains URIs as the dataset is created from DBPedia (An automatically generated RDF database from Wikipedia metadata)[30] entries. To adapt the dataset to work with the RDF graph generated by GraphGen, we modified the vquanda dataset to only preserve the last part of the URI in the triples section of SPARQL queries (Fig. 16). We also flipped the content so that the pattern matching section of the query would precede the command, so that Keyword Enforce can provide important context before declaring variables, for example, a COUNT() command would not be viable when asked to get the population of a city, as the triple store stores the population size as a single node instead of representing every individual with a separate node.

Original SPARQL	`SELECT DISTINCT COUNT(?uri) WHERE { ?x <http://dbpedia.org/ontology/commander> <http://dbpedia.org/resource/Andrew_Jackson> . ?uri <http://dbpedia.org/ontology/knownFor> ?x . }`
Modified SPARQL	`{ ?x <commander> <Andrew_Jackson> . ?uri <knownFor> ?x . } SELECT DISTINCT COUNT(?uri) WHERE`

Fig. 16: Comparison of the SPARQL in the original dataset and the modified dataset

Using this modified dataset, we fine-tuned an unsloth optimized Gemma2-2b model with a total batch size of 8 and fine-tuned for 600 steps, and a Gemma2-9b variant finetuned with a batch size of 8 for 1000 steps.

SPARQL Enforce & Keyword Enforce

QueryGen combines the language model with a custom logit processor to ensure it generates valid SPARQL queries that works with the generated RDF triple store.

SPARQL, like RDF triples, also relies on the subject-predicate-object structure. Rules are added as part of the custom logit processor to enforce the overall SPARQL structure: Brackets needs to be closed, etc. It uses the same algorithm as TRSLogit to detect the nearest special token.

Because the SPARQL queries against the RDF database generated by GraphGen, the SPARQL must use keywords that are relevant to the generated RDF database. We enforce this by limiting the model to only selecting from existing keywords for non-variables similar to Keyword Reward in GraphGen, however, instead of rewarding existing tokens based on tree, Keyword Enforce disables all tokens that does not match the tree or SPARQL Enforce. The logit processor detects if the predicted token is a part of a triple keyword, then based on the position of the keyword in a triple it will limit the allowed tokens to either the node tokens or the predicate tokens and certain special tokens.

Special token \ Token position	tokens exist between currently predicting and special token	Special token is directly before currently predicting
	Only allow these	Only allow these
{	IMPOSSIBLE	< ?
<	keyword(node/predicate) >	keyword(node/predicate)
>	IMPOSSIBLE	<(index<2) ?(index<2) .(index==2)
.	IMPOSSIBLE	< ? }
?	ALL, except > { }(index<2) .(index<2) ?(index==2) <(index==2)	ALL, except < > { } ? .
}	IMPOSSIBLE	<eos>
None	ALL	ALL

Fig. 17: Rule table for allowed tokens based on closest special token

Fig. 17 shows the rules the custom logits processor uses to enforce the SPARQL structure and the use of existing keywords. The allowed keywords change based on the position of the keyword. In position 0 and 2 node keywords are allowed, in position 1 predicate keywords are allowed. Variables do not have this keyword restriction as they are defined in the SPARQL query. This allows QueryGen to generate contextual SPARQL queries based on user's questions.

3.3. AnswerGen

The final part of the SemanticLang QA system is the answer contextualization module, AnswerGen. RDF databases, when encountered a SELECT query, typically reply with a list of keywords that satisfy the conditions. These answers, while coherent, do not provide the necessary context. AnswerGen uses the user's original question and the query result to provide a coherent response. It is a Gemma2-2b language model fine-tuned with the vquanda dataset as shown in Fig. 18. It is trained with a total batch size of 8 for 300 steps.

question	output	answer
Count the number of people became famous for when Andrew Jackson was a commander ?	8	There are 8 people known for works commanded by Andrew Jackson.

Fig. 18: An entry in the vquanda answer contextualization dataset

AnswerGen can incorporate the query result into a natural sentence as a reply to the user's question, completing the QA system.

The combination of GraphGen for document understanding, QueryGen for question understanding and AnswerGen for answer contextualization completes the seq2seq

document QA system SemanticLang. A complete QA run is shown in Fig. 19.

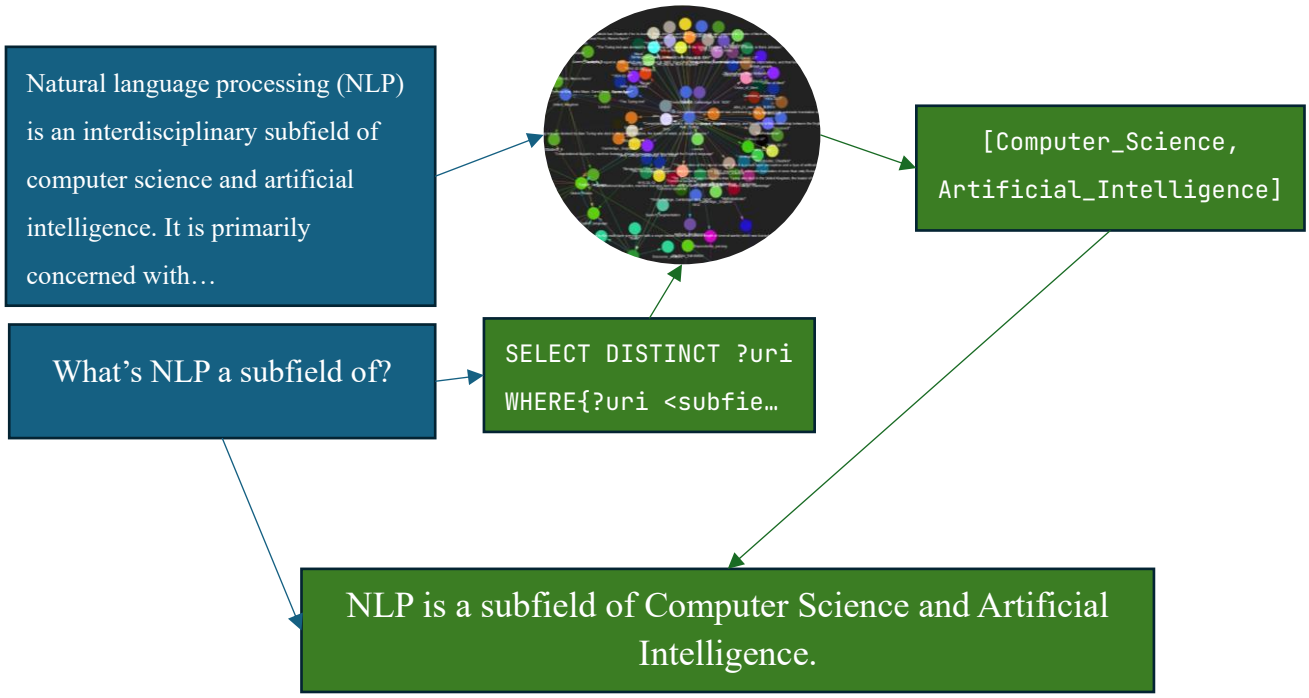

Fig. 19: A demonstration of SemanticLang's entire QA process.

After the answer is generated, the system can backtrack to specific text chunks used for answering. By modifying QueryGen's SPARQL command to use CONSTRUCT instead of SELECT, a subgraph of nodes used for deduction can be generated. By using the document and text chunk ID assigned to each node by GraphGen, the specific chunks of text used to derive the answer can be compiled and presented to the user as evidence. Compared to entire documents, text chunks have a shorter length, therefore making it easier for users to pinpoint the specific information in the evidence text the model used.

4. Performance Evaluation

This section evaluates the GraphGen module's knowledge extraction ability.

> Artificial intelligence (AI), in its broadest sense, is intelligence exhibited by machines, particularly computer systems.
>
> […]
>
> they are unsafe, and the use of self-learning neural networks trained on vast, unregulated sources of flawed internet data should be curtailed.[dubious – discuss][213]
>
> ***(https://en.wikipedia.org/wiki/Artificial_intelligence), 45890 characters***

Fig. 20: A long document.

We found that GraphGen has the unique ability of being able to process long documents. For example, the document shown in Fig. 19 has a character length of 6725 words or 45890 characters (8946 tokens for GPT-4[31], 9407 tokens for Gemma2[17]), Gemma2 and GPT-4 has a context length of 8192 tokens[17], [31], which is below the number of tokens for this document. GraphGen, however, can understand documents at this length.

We used WebNLG test dataset to compare the performance of GraphGen2b/9b with Gemini-1.5-Flash[32] and Gemma2-2b/9b Instruction-tuned model on medium-length sentence level relation extraction. We calculate the sum of squared differences between Laplacian spectrum of the predicted RDF graph and the ground truth to generate a loss score (Fig. 21).

```python
def select_k(spectrum, minimum_energy = 0.9):
    running_total = 0.0
    total = sum(spectrum)
    if total == 0.0:
        return len(spectrum)
    for i in range(len(spectrum)):
        running_total += spectrum[i]
        if running_total / total >= minimum_energy:
            return i + 1
    return len(spectrum)

# Similarity
laplacian1 = nx.spectrum.laplacian_spectrum(ground_truth_graph)
laplacian2 = nx.spectrum.laplacian_spectrum(predicted_graph)

k1 = select_k(laplacian1)
k2 = select_k(laplacian2)
k = min(k1, k2)

similarity = sum((laplacian1[:k] - laplacian2[:k])**2)
```

Fig. 21: Algorithm for calculating prediction accuracy

The resulting score is a positive number, where closer to 0 means higher accuracy.

WebNLG loss (Lower is better)	**GraphGen 2B**	**GraphGen 9B**	Gemini-1.5-Flash	Gemma2-2b-it	Gemma2-9b-it
Mean	**4.8168**	**4.5607**	3.3567	5.9007	4.6123
Median	**1.0000**	**0.7639**	0.0000	4.0000	1.0000

We compared SemanticLang's QA ability with Gemini-1.5-Flash and Gemma2-2b/9b instruction tuned models using the Stanford Question Answering Dataset 2.0(SQuAD2.0)[33] dev dataset. An exact match score (1 for matching ground truth, else 0) and F1 score is calculated for each question. The algorithm used for calculating scores is from

SQuAD2.0 score (Higher is better)	**Semantic-Lang 2B**	**Semantic-Lang 9B**	Gemini-1.5-Flash	Gemma2-2b-it	Gemma2-9b-it
EM Mean	**0.4031**	**0.4318**	0.3982	0.3564	0.3900
F1 Mean	**0.4030**	**0.4212**	0.4398	0.4019	0.4321

We find that SemanticLang not only has lower loss in knowledge extraction compared to Gemma2 local lightweight LLMs, SemanticLang manages to surpass Gemini-1.5-Flash in average Exact Match score on the SQuAD2.0 dev dataset. We conclude that

SemanticLang can successfully extract information and answer users' questions with reasonable accuracy and reliability, while also having the unique capabilities of infinite context length, knowledge visualization and sentence-level evidence backtracking for explainable deduction.

5. Conclusion and Future Work

This paper proposes a lightweight document knowledge extraction and QA model designed to have human-verifiable logic deduction processes. To achieve this, we created GraphGen, QueryGen and AnswerGen that combined neural language models with symbolic RDF graphs to improve the logic stability and transparency of the system. In GraphGen, we propose Multi Context to enable the model to understand long documents, improve performance and improve explainability enabled by the special properties of RDF graphs. We created an explainable AI that can be run locally on-device for safe processing of private information and low power usage, it also provides sentence-level evidence for its answers, so users can examine and verify the model's understanding, logic deduction and answering process, the generated RDF graph can also be visualized to aid the user in exploring knowledge presented in the text, or to be integrated into other systems.

While SemanticLang has demonstrated its verifiable text understanding and deduction abilities, we will continue the development on SemanticLang to further improve its reliability and to add features. For example, GraphGen sometimes generates unnecessarily detailed triple keywords, which can reduce the chance of a successful query. We also wish to implement parallelization to GraphGen to improve its performance, and to enable QueryGen to understand more nuanced context from the RDF graph than just keywords.

In conclusion, we believe that SemanticLang can be a starting point for reliable evidence-first user-supervisable language models.

6. References

[1] T. B. Brown *et al.*, “Language Models are Few-Shot Learners,” Jul. 22, 2020, *arXiv*: arXiv:2005.14165. doi: 10.48550/arXiv.2005.14165.

[2] OpenAI *et al.*, “GPT-4 Technical Report,” Mar. 04, 2024, *arXiv*: arXiv:2303.08774. doi: 10.48550/arXiv.2303.08774.

[3] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, “The Curious Case of Neural Text Degeneration,” Feb. 14, 2020, *arXiv*: arXiv:1904.09751. doi: 10.48550/arXiv.1904.09751.

[4] A. Vaswani *et al.*, “Attention Is All You Need,” Aug. 01, 2023, *arXiv*: arXiv:1706.03762. Accessed: Jul. 20, 2024. [Online]. Available: http://arxiv.org/abs/1706.03762

[5] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, “Efficient Transformers: A Survey,” Mar. 14, 2022, *arXiv*: arXiv:2009.06732. doi: 10.48550/arXiv.2009.06732.

[6] T. Niven and H.-Y. Kao, “Probing Neural Network Comprehension of Natural Language Arguments,” Sep. 16, 2019, *arXiv*: arXiv:1907.07355. doi: 10.48550/arXiv.1907.07355.

[7] S. Jain and B. C. Wallace, “Attention is not Explanation,” May 08, 2019, *arXiv*: arXiv:1902.10186. doi: 10.48550/arXiv.1902.10186.

[8] M. Shanahan, “Talking About Large Language Models,” Feb. 16, 2023, *arXiv*: arXiv:2212.03551. doi: 10.48550/arXiv.2212.03551.

[9] J. Wei *et al.*, “Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,” Jan. 10, 2023, *arXiv*: arXiv:2201.11903. doi: 10.48550/arXiv.2201.11903.

[10] D. Zhou *et al.*, “Least-to-Most Prompting Enables Complex Reasoning in Large Language Models,” Apr. 16, 2023, *arXiv*: arXiv:2205.10625. doi: 10.48550/arXiv.2205.10625.

[11] Z. Ji *et al.*, “Survey of Hallucination in Natural Language Generation,” *ACM Comput Surv*, vol. 55, no. 12, p. 248:1-248:38, Mar. 2023, doi: 10.1145/3571730.

[12] L. Pan, A. Albalak, X. Wang, and W. Y. Wang, “Logic-LM: Empowering Large Language Models with Symbolic Solvers for Faithful Logical Reasoning,” Oct. 18, 2023, *arXiv*: arXiv:2305.12295. Accessed: Jul. 19, 2024. [Online]. Available: http://arxiv.org/abs/2305.12295

[13] S. Han *et al.*, “FOLIO: Natural Language Reasoning with First-Order Logic,” May 17, 2024, *arXiv*: arXiv:2209.00840. Accessed: Aug. 01, 2024. [Online]. Available: http://arxiv.org/abs/2209.00840

[14] “W3C Semantic Web Activity Homepage.” Accessed: Sep. 15, 2024. [Online]. Available: https://www.w3.org/2001/sw/

[15] “Resource Description Framework (RDF) Model and Syntax Specification.” Accessed: Sep. 11, 2024. [Online]. Available: https://www.w3.org/TR/PR-rdf-syntax/

[16] “SPARQL 1.1 Overview.” Accessed: Sep. 15, 2024. [Online]. Available: https://www.w3.org/TR/sparql11-overview/

[17] Gemma Team *et al.*, “Gemma 2: Improving Open Language Models at a Practical Size,” Aug. 02, 2024, *arXiv*: arXiv:2408.00118. doi: 10.48550/arXiv.2408.00118.

[18] “GPT-4o System Card.” Accessed: Sep. 15, 2024. [Online]. Available: https://openai.com/index/gpt-4o-system-card/

[19] “World Wide Web - Wikipedia.” Accessed: Sep. 15, 2024. [Online]. Available: https://en.wikipedia.org/wiki/World_Wide_Web

[20] “Semantic Web,” *Wikipedia*. Aug. 30, 2024. Accessed: Sep. 15, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Semantic_Web&oldid=1243036247

[21] “Resource Description Framework,” *Wikipedia*. Sep. 03, 2024. Accessed: Sep. 15, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Resource_Description_Framework&oldid=1243769359

[22] *unslothai/unsloth*. (Sep. 15, 2024). Python. Unsloth AI. Accessed: Sep. 15, 2024. [Online]. Available: https://github.com/unslothai/unsloth

[23] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini, “Creating Training Corpora for NLG Micro-Planners,” in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, R. Barzilay and M.-Y. Kan, Eds., Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 179–188. doi: 10.18653/v1/P17-1017.

[24] E. Loper and S. Bird, “NLTK: The Natural Language Toolkit,” May 17, 2002, *arXiv*: arXiv:cs/0205028. doi: 10.48550/arXiv.cs/0205028.

[25] “Quiz bowl,” *Wikipedia*. Jul. 24, 2024. Accessed: Sep. 15, 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Quiz_bowl&oldid=1236341758

[26] T. Kudo and J. Richardson, “SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing,” Aug. 19, 2018, *arXiv*: arXiv:1808.06226. doi: 10.48550/arXiv.1808.06226.

[27] “Tools to Manipulate and Query Semantic Data.” Accessed: Sep. 15, 2024. [Online]. Available: https://docs.ropensci.org/rdflib/

[28] “WestHealth/pyvis: Python package for creating and visualizing interactive network graphs.” Accessed: Sep. 15, 2024. [Online]. Available: https://github.com/WestHealth/pyvis

[29] E. Kacupaj, H. Zafar, J. Lehmann, and M. Maleshkova, “VQuAnDa: Verbalization QUestion ANswering DAtaset,” in *The Semantic Web*, A. Harth, S. Kirrane, A.-C. Ngonga Ngomo, H. Paulheim, A. Rula, A. L. Gentile, P. Haase, and M. Cochez, Eds., Cham: Springer International Publishing, 2020, pp. 531–547. doi: 10.1007/978-3-030-49461-2_31.

[30] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, “DBpedia: A Nucleus for a Web of Open Data,” in *The Semantic Web*, K. Aberer, K.-S. Choi, N. Noy, D. Allemang, K.-I. Lee, L. Nixon, J. Golbeck, P. Mika, D. Maynard, R. Mizoguchi, G. Schreiber, and P. Cudré-Mauroux, Eds., Berlin, Heidelberg: Springer, 2007, pp. 722–735. doi: 10.1007/978-3-540-76298-0_52.

[31] “Snapshot.” Accessed: Sep. 15, 2024. [Online]. Available: https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4

[32] “[2403.05530] Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context.” Accessed: Nov. 24, 2024. [Online]. Available: https://arxiv.org/abs/2403.05530

[33] “[1806.03822] Know What You Don’t Know: Unanswerable Questions for SQuAD.” Accessed: Nov. 24, 2024. [Online]. Available: https://arxiv.org/abs/1806.03822

7. Acknowledgements

This project was inspired by recent controversies online surrounding commercial Large Language Models, specifically controversies over copyright in the training data, power usage and privacy concerns over server-side inference and the reliance some people have over LLM's results because of LLM's confident wording. This project aims to create a lightweight AI system that acts similarly to conventional chat-based LLMs but can be run locally and have a verifiable logic deduction process. The original project proposed combining LLMs with predicate logic solvers to enhance the model's reasoning process. However, during research, I found that most NL predicate logic datasets focus on pure logic questions. Ms. Di Wu suggested to me that the model should focus on semantics instead of pure logic, as semantic relations can provide more useful information for common question-answering tasks like those of search engines and typical commercial LLM usage. From Ms. Di Wu's suggestion I discovered the Semantic Web and the Resource Description Framework which became the key to this project. Thanks to Ms. Di Wu for helping me design the proper direction of the project during multiple stages of the project. Thanks to Prof. DongDong Weng at the Beijing Institute of Technology, who gave me valuable guidance on the algorithm design and logit processors of the project. I would also like to thank Mr. YiNing Shi for his guidance and help during this project, and for introducing me to Prof. DongDong Weng. Ms. Di Wu and Mr. YiNing Shi are teachers at the High School affiliated to the Renmin's University of China. I'm the sole researcher, system designer, developer, and paper writer for this project. Multiple challenges were faced during the development of this project. For example, the GraphGen module would produce incorrect triples formats, which we fixed by developing TRSLogits. I would like to again thank Ms. Di Wu, Prof. DongDong Weng and Mr. YiNing Shi for their unpaid guidance in this project.