I NanoSocrates: Foundational Models, Building a Very Small Semantic Language Model

1 Task Overview

The field of Natural Language Processing is increasingly dominated by Foundational Models, large-scale models pre-trained on vast amounts of data that can be adapted to a wide range of downstream tasks. This project challenges students to delve into the core principles of these models by building a **Very Small Semantic Language Model** from scratch. The goal is to develop a single, unified Transformer-based model capable of understanding and translating between unstructured natural language text and structured data in the form of Resource Description Framework (RDF) triples. The model, which we will call **NanoSocrates**, will be trained in the domain of movies and must perform four distinct but related tasks:

1. **Text-to-RDF (Text2RDF) - RDF Generation on the Decoder side:** Converting a natural language text into a set of RDF triples that capture its semantic meaning.
2. **RDF-to-Text (RDF2Text) - Text Generation on the Decoder side:** Generating a coherent, human-readable sentence from a set of RDF triples.
3. **RDF Completion 1 - Masked Language Modeling with Spanned Masking:** Predicting one or more missing components (subject, predicate, or object) within RDF triples originally substituted with `<MASK>` tokens, a task analogous to link prediction in knowledge graphs.
4. **RDF Completion 2 - RDF Generation on the Decoder side:** Predicting one or more missing RDF triples (subject, predicate, or object) after a set of given triples, a task analogous to knowledge completion in knowledge graphs.

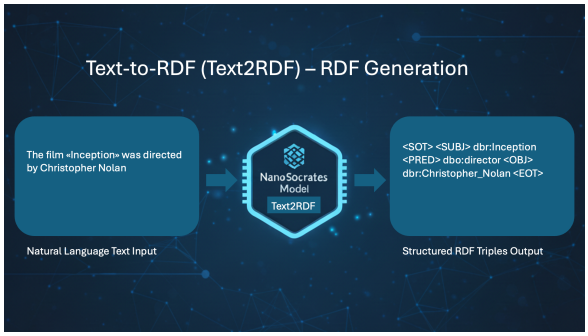
This project provides hands-on experience in model architecture design, large-scale data curation, custom tokenizer development, and multi-task pre-training, key skills required to build modern AI systems.

2 Objective

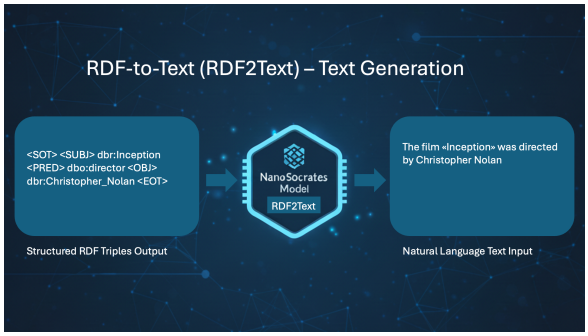
The main objective is to design, pre-train, and evaluate a novel Encoder-Decoder Transformer model specialized for bidirectional text and knowledge graph translation.

Sub-Objectives

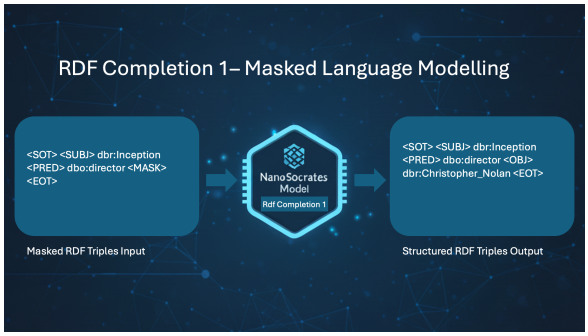
- **Data Curation:** Collect and align a large-scale dataset for the movie domain by pairing Wikipedia article texts with the corresponding RDF triples from DBpedia.
- **Tokenization:** Design and train a custom Byte-Pair Encoding (BPE) tokenizer on the curated corpus, including specialized tokens required to handle the structured nature of RDF data.
- **Model Implementation:** Implement an Encoder-Decoder Transformer architecture, inspired by models like T5, optimized for sequence-to-sequence tasks.



(a) Text-to-RDF (Text2RDF): Generating structured RDF triples from natural language text.



(b) RDF-to-Text (RDF2Text): Generating a coherent sentence from a set of RDF triples.



(c) RDF Completion 1 (Masked Language Modeling): Predicting a masked component within an RDF triple.

(d) RDF Completion 2 (RDF Generation): Predicting subsequent triples based on a given context.

Figure 1: The four primary tasks of the **NanoSocrates** model. These tasks demonstrate the model's unified capability to translate between unstructured natural language and structured knowledge, as well as perform knowledge graph completion. Background generated with NanoBanana.

- **Pre-training:** Pre-train the model on a mixture of the four core tasks (Text2RDF, RDF2Text, RDF Completion 1 and 2) to create a single, versatile semantic model.
- **Evaluation:** Rigorously evaluate the model's performance on each of the four tasks using appropriate and distinct metrics.

The main goal is to implement a multi-stage computer vision pipeline that can index a labeled dataset of objects (characters) and use this database to identify and classify objects within new, complex scenes.

3 Dataset

For this project, you will create a dataset by combining information from two major public knowledge sources: **Wikipedia** and **DBpedia**. DBpedia is a crowd-sourced community effort to extract structured content from the information created in Wikipedia.
The process involves:

- **Entity Sourcing:** Programmatically identify a large list of movie entities from DBpedia. You can use the DBpedia SPARQL endpoint (`https://dbpedia.org/sparql`) to query for resources of type `dbo:Film` and at least its one-hop neighbors in the knowledge graph.
- **Data Collection:** For each movie entity:
  - Retrieve all associated RDF triples from DBpedia.
  - Fetch the introductory text (abstract) of the corresponding English Wikipedia page.
- **Dataset Creation:** Create a final dataset consisting of paired (text, triples) examples. This will serve as the basis for training all four tasks.

4 Data Preprocessing and Tokenization

A critical step in this project is converting the structured RDF graph data and the task directives into a linear sequence format that the Transformer can process.

- **RDF Serialization:** RDF triples must be serialized into a string. You will define a format using special tokens. For example: `<SOT>` `<SUBJ>` dbr:Inception `<PRED>` dbo:director `<OBJ>` dbr:Christopher_Nolan `<EOT>`
- **Task Formatting:** Each input sequence must be prefixed with a special token to instruct the model on the task to perform.
  - **Text2RDF Input:** The film 'Inception' was directed by Christopher Nolan. `<Text2RDF>`
  - **RDF2Text Input:** `<SOT>` `<SUBJ>` dbr:Inception `<PRED>` dbo:director `<OBJ>` dbr:Christopher_Nolan `<EOT>` `<RDF2Text>`
  - **RDF Completion 1 (Masked Language Modeling - Encoder Side) Input:** `<SOT>` `<SUBJ>` dbr:Inception `<PRED>` dbo:director `<MASK>` `<EOT>`
  - **RDF Completion 2 (Text/RDF Generation - Decoder Side) Input:** `<SOT>` `<SUBJ>` dbr:Inception `<PRED>` dbo:director `<OBJ>` dbr:Christopher_Nolan `<EOT>` `<CONTINUERDF>`
- **Custom Tokenizer:** You must train a new tokenizer (e.g., BPE) from scratch on your final curated dataset (a mix of natural language text and linearized RDF). This ensures that both vocabularies are represented efficiently.
- **Special Tokens:** The tokenizer's vocabulary must be augmented with all necessary special tokens, including: `<SOT>` (Start of Triple), `<EOT>` (End of Triple), `<SUBJ>`, `<PRED>`, `<OBJ>`, `<RDF2Text>`, `<Text2RDF>`, `<CONTINUERDF>`, and `<MASK>`.

5 System Architecture

The core of this project is the implementation of the NanoSocrates model.

- **Base Architecture:** The model must be an Encoder-Decoder Transformer.
  - The Encoder is responsible for creating a rich contextual representation of the input sequence (whether it is text or linearized RDF).

  - The Decoder is responsible for generating the target sequence autoregressively, conditioned on the encoder's output.

- **Attention Mechanism:** The base model should be built using standard Multi-Head Self-Attention layers. Students are encouraged to research and discuss more efficient attention variants, such as **Multihead Latent Attention (MLA)**, and those using **Rotary Position Embeddings (RoPE)**, and **Interleaved Attention layers**, which can improve performance.
- **Masked Language Modeling (MLM) / Spanned Masking:** The students are encouraged to integrate the **Spanned Masking technique** to handle entities with varying length representations.

6 Hints

To make this project manageable within a one-month timeframe and with limited computational resources, consider the following strategies. The goal is to simplify the scope without sacrificing the core learning objectives.
You can:

- **Drastically Reduce the Dataset Size:**
  - For each movie, use only the **first paragraph** of the Wikipedia abstract rather than the entire text.
  - Filter the DBpedia triples to keep only the **most common and informative predicates**. This simplifies the RDF structure and reduces the vocabulary size for the tokenizer.
- **Keep the Model Architecture Small:**
  - **Implement a "micro" Transformer.** A model with 2-4 encoder layers and 2-4 decoder layers is a reasonable starting point.
  - **Use a small hidden dimension** (e.g., d_model = 256 or 512) and a reduced number of attention heads (e.g., 4 or 8).
  - **Limit the maximum sequence length** for both the encoder and decoder (e.g., 256 or 512 tokens). Truncate any examples that are longer. This is the single most effective way to reduce memory usage and training time.
- **Adopt a Smart Training and Debugging Workflow:**
  - **Start with a toy dataset:** Before running the full pipeline, create a tiny dataset of just 10-20 movies. Get your entire workflow (data processing, tokenization, training, and evaluation) working end-to-end on this small dataset.
  - **Overfit a single batch:** As a sanity check, ensure your model can achieve near-zero loss when trained repeatedly on a single batch of data. This confirms that the model architecture and training loop are correctly implemented.
  - **Use a smaller vocabulary:** When training your tokenizer, you can limit the vocabulary size (e.g., to 16,000 or 32,000 tokens) to keep the model's embedding layer small and memory-efficient.

7 Training and Evaluation

Training

- Implement a multi-task training loop. Each batch should contain a mix of examples from the Text2RDF, RDF2Text, and RDF Completion tasks.
- Use the AdamW optimizer with a suitable learning rate scheduler.

Evaluation

- The model performance must be evaluated separately for each task using appropriate metrics:
  - **RDF2Text:** Use standard text-generation metrics such as **BLEU**, **ROUGE**, and **METEOR**.
  - **Text2RDF:** Evaluate the generated triples against the ground truth using **Precision**, **Recall**, and **F1-score**.
  - **RDF Completion 1:** Measure the **Accuracy** of predicting the correct masked entity or predicate.
  - **RDF Completion 2:** Evaluate the generated triples against the ground truth using **Precision**, **Recall**, and **F1-score**.

8 Model Analysis

Perform an analysis of the model's performance. Identify any challenges or limitations faced by the model. Provide recommendations for further improvements or modifications to enhance the model's accuracy and efficiency.

9 Deliverables

- A well-commented Python code implementation of the models.
- A report documenting the detailed approach taken to solve the problem, including data preprocessing, model architecture, training, and evaluation results.
- Analysis and discussion of the model's performance, challenges faced, and recommendations for improvement.
- A presentation that will be discussed at the time of the Oral Exam.

Notes

You are encouraged to utilize existing deep learning libraries such as TensorFlow or PyTorch for implementing the model. Make sure to properly document your code and provide clear explanations in the report to demonstrate your understanding of the concepts and techniques used.