| CS 391L | Machine Learning WB | Spring 2025 |
|---|---|---|

# Final Project Ideas

Lecturer: Inderjit Dhillon Date: Mar 24, 2025

## Project 1: Exploring and Interpreting Attention Patterns in Needle-in-a-Haystack Tasks

**Overview:** This project investigates how transformer-based language models use attention to find a small, targeted piece of information (the "needle") hidden inside a large amount of irrelevant text (the "haystack"). This task is a key benchmark for testing a model's ability to retrieve specific information from within its context window [Goo2x, Kam2x, Tow2x]. Recent models like Google's Gemini 1.5 Pro have shown nearly perfect recall even with context windows of up to one million tokens across text, video, and audio [Goo2x]. However, research indicates that retrieval performance can drop when multiple needles are embedded or when the needle is placed at certain positions within the context [Goo2x, Tow2x].

**Objectives:**

- Implement the needle-in-a-haystack task using a transformer-based model.
- Visualize attention patterns from various layers and heads for different context lengths and needle positions.
- Analyze the relationship between attention weights and successful needle retrieval.
- Study the effect of multiple needles on attention patterns and retrieval accuracy.
- Compare attention behaviors across different model architectures (if feasible).

**Potential Methodology:** A potential approach involves setting up the needle-in-a-haystack task using tools such as the `needlehaystack` package [Kam2x] or a custom implementation. Experiments can be performed using the Python `transformers` library to access and inspect the attention matrices during model inference. Visualization techniques, such as heatmaps and network graphs, can be employed to illustrate which parts of the input the model focuses on [CVM19, 3Bl2x]. Experiments can be designed to systematically vary the length of the haystack, the position of the needle (e.g., beginning, middle, end), and the number of needles. Quantitative analysis may involve comparing the average attention weight on the needle against that on the surrounding haystack, while qualitative analysis can involve visually inspecting the attention maps to identify potential patterns or biases in how the model attends to the relevant information.

## Project 2: Building a Retrieval-Augmented Generation (RAG) System for Course Material Question Answering

**Description:** This project explores the development of a question answering system based on the Retrieval-Augmented Generation (RAG) framework, which integrates external course material into the response generation process. RAG systems can enhance large language models (LLMs) by allowing them to access domain-specific information that may not be fully covered during training [Coh2x, Dat2xb, K2v2x, Med2xc, dee2xb, Int2x, IBM2x].

Grounding LLM responses in retrieved, relevant information can help improve accuracy and reduce the risk of generating factually incorrect or nonsensical answers. A typical RAG system includes several components: a document store for course materials, a chunker to segment documents, an embedder to create vector representations, a retriever to identify the most relevant segments, a prompt builder to combine the user query with the retrieved context, and the LLM to generate the final answer [Wil2x, Pro2xb, Ton2x, Med2xb, Dja2x, Tim2x]. Open-source libraries and platforms such as Langchain and deepset AI Platform offer useful tools to simplify the process of building and deploying RAG systems [dee2xb, Dja2x, Lan2x, Dat2xa].

**Potential Objectives:**

- Process and index the provided course materials for efficient retrieval.
- Implement a retrieval mechanism to locate relevant content based on user queries.
- Construct effective prompts by combining the user query with the retrieved context for the LLM.
- Utilize a chosen LLM to generate answers based on the retrieved information.
- Evaluate the performance of the RAG system using appropriate metrics.
- Explore different retrieval strategies or LLM configurations to potentially optimize performance.

**Potential Methodology:** A potential approach can begin with preprocessing the course materials (e.g., lecture notes, recordings, assignments) by converting documents into plain text. Experiments can be performed to implement a chunking strategy that divides the text into smaller, semantically coherent segments [K2v2x, Wil2x, Dja2x]. Next, an embedding model can be selected to create vector representations of these text chunks, which can then be indexed in a vector store for efficient similarity search [Wil2x, Dja2x, Tim2x]. When a user poses a question, the same embedding model can be applied to generate a vector representation of the query, and a retrieval mechanism can be used to fetch the top-k most relevant text chunks based on semantic similarity [Pro2xb, Ton2x, Med2xb, Dja2x]. These retrieved segments may then be combined with the original query to construct a prompt for the LLM [Ton2x, Tim2x], which in turn generates an answer based on the provided context [Ton2x, Med2xb, Dja2x, Tim2x]. Finally, the system's performance can be evaluated using metrics such as retrieval precision, recall, and measures of answer relevance and faithfulness. Additionally, experiments can explore different retrieval algorithms, embedding models, prompt templates, and LLM configurations to assess their impact on overall system performance.

## Project 3: Exploring Tokenization Strategies for C++ Code LM

**Description:** This project explores various tokenization strategies specifically designed for C++ code language model (LM). Tokenization—the process of breaking a continuous stream of characters into meaningful units (tokens)—is a crucial first step in many text and code processing pipelines [Sta2x, Gee2x, Ari2x, Med2xa, arX2x]. The way C++ code is tokenized can significantly affect downstream training of machine learning models on code [Ari2x, Med2xa, arX2x]. C++ presents unique challenges for tokenization due to its complex grammar, which includes overloaded operators (e.g., `>>` that can denote a right-shift or the closing of nested templates), preprocessor directives, and intricate syntax rules [Sta2x, Res2x]. Basic tokenization methods based on whitespace and delimiters might not suffice and could lead to ambiguities or mis-parsed code. More advanced tokenization techniques, common in natural language processing for handling rare words and out-of-vocabulary terms, might be adapted to improve the handling of long C++ identifiers or uncommon coding patterns [Res2x]. Given the impact that tokenization choices have on the representation and subsequent processing of C++ code.

**Potential Objectives:**

- Implement and compare various tokenization strategies for C++ code.
- Analyze the tokens generated by each strategy on a diverse set of C++ code samples.
- Investigate how each strategy handles complex C++ syntax, including operators, templates, and preprocessor directives.
- Evaluate the impact of different tokenization strategies on downstream performance in language model tasks.

**Potential Methodology:** A suggested approach is to first collect a diverse set of C++ source code samples that represent various coding styles and language features. One may then implement several tokenization methods—ranging from simple regular expression-based techniques [Gee2x] to more sophisticated methods that mimic the approaches used by C++ compilers or some adaptations of sub-word tokenization algorithms (e.g., Byte-Pair Encoding or WordPiece). The effectiveness of these methods can be compared using several metrics, such as the total number of tokens generated, the accuracy in identifying and separating syntactic elements (e.g., operators, keywords, identifiers, literals), and the handling of complex constructs like nested templates or preprocessor directives [Sta2x]. A qualitative analysis of the token sequences may help assess their granularity and correctness—for example, whether multi-character operators like `>>` are treated as a single token or split into two. Finally, training a small language model using each tokenization strategy and evaluating its performance on simple C++ code tasks could provide insights into the downstream impact of the tokenization choice.

## Project 4: Designing and Implementing Evaluation Metrics for Assessing Hallucinations in LLM-Generated Text

**Description:** This project addresses the challenge of hallucinations in large language models (LLMs)—instances where the generated text, despite appearing coherent and plausible, contains factual errors or fabricated information. Hallucinations can undermine the reliability of LLM outputs, particularly in critical applications. Therefore, it is important to develop methods that can accurately evaluate and quantify hallucinations. The project will begin with a review of existing evaluation metrics that detect hallucinations in LLM-generated text. These techniques include analyzing log probabilities of generated sequences, measuring semantic similarity against a ground truth, employing automated fact verification against external knowledge bases, and even using other LLMs as evaluators [Fid2x, Dee2xa, Pro2xa, Ama2x, AWS2x, Con2x]. Based on this literature review, the project can either propose novel evaluation metrics or combine existing ones to provide a more nuanced assessment of hallucinations. The proposed metrics can focus on more nuanced aspects like consistency with provided context and grounding in external sources.

**Potential Objectives:**

- Research and analyze existing evaluation metrics for hallucinations in LLM-generated text.
- Design one or more evaluation metrics tailored for specific types of LLM output (e.g., question answering, summarization).
- Evaluate the hallucination rate of one or more LLMs on a chosen dataset using the implemented metrics.
- Compare the performance of the newly implemented metrics with established ones.
- Explore potential correlations between different metrics and human evaluations of text quality and factuality.

**Potential Methodology:** A potential approach could start with a review of the current literature on hallucination evaluation in LLMs. One might select a specific natural language processing task—such as question answering or text summarization—and obtain or curate a dataset with corresponding ground truth information. Experiments can then be performed by implementing several existing hallucination detection techniques, such as:

- Calculating log probabilities of generated text.
- Assessing semantic similarity between generated text and ground truth (e.g., using cosine similarity).
- Employing automated fact-checking methods.

Based on these evaluations try to propose either modified or a new metric for hallucination detection. After implementing the metrics, one can run selected LLMs on the dataset and use the new metrics to quantify hallucinations. Finally, the results can be compared with those obtained from established metrics, and further analysis may explore the correlation between metric scores and human judgments of quality and factuality.

## References

[3Bl2x] 3Blue1Brown. Visualizing attention, a transformer's heart. https://www.3blue1brown.com/lessons/attention, 202x. Video.

[Ama2x] Amazon Science Blog. New tool, dataset help detect hallucinations in large language models. https://www.amazon.science/blog/new-tool-dataset-help-detect-hallucinations-in-large-language-models, 202x.

[Ari2x] Arize AI Blog. Tokenization. https://arize.com/blog-course/tokenization/, 202x.

[arX2x] arXiv. The impact of tokenization on large language models. https://arxiv.org/html/2402.01035v2, 202x.

[AWS2x] AWS Machine Learning Blog. Reducing hallucinations in large language models with custom intervention using amazon bedrock agents. https://aws.amazon.com/blogs/machine-learning/reducing-hallucinations-in-large-language-models-with-custom-intervention-using-amazon-bedr 202x.

[Coh2x] Cohere Blog. Rag architecture. https://cohere.com/blog/rag-architecture, 202x.

[Con2x] Confident AI Docs. Metrics - hallucination. https://docs.confident-ai.com/docs/metrics-hallucination, 202x.

[CVM19] Kevin Clark, Elena Voita, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

[Dat2xa] Data Science Toolkit. Question answering using retrieval augmented generation (rag + q&a). https://www.ds-toolkit.com/assets/533862c7-ae3e-4338-9fe1-0e45fe50b436, 202x.

[Dat2xb] Databricks Glossary. Retrieval augmented generation (rag). https://www.databricks.com/glossary/retrieval-augmented-generation-rag, 202x.

[Dee2xa] Deepchecks Blog. Llm hallucination detection and mitigation. https://www.deepchecks.com/llm-hallucination-detection-and-mitigation-best-techniques/, 202x.

[dee2xb] deepset AI Docs. Retrieval augmented generation (rag) question answering. https://docs.cloud.deepset.ai/docs/generative-question-answering, 202x.

[Dja2x] Django Stars Blog. Rag question answering with python. https://djangostars.com/blog/rag-question-answering-with-python/, 202x.

[Fid2x] Fiddler AI Blog. Detect hallucinations using llm metrics. https://www.fiddler.ai/blog/detect-hallucinations-using-llm-metrics, 202x.

[Gee2x] GeeksforGeeks. Tokenizing a string in c++. https://www.geeksforgeeks.org/tokenizing-a-string-cpp/, 202x.

[Goo2x] Google Cloud. The needle in the haystack test and how gemini pro solves it. https://cloud.google.com/blog/products/ai-machine-learning/the-needle-in-the-haystack-test-and-how-gemini-pro-solves-it, 202x.

[IBM2x] IBM Research Blog. What is retrieval-augmented generation? https://research.ibm.com/blog/retrieval-augmented-generation-RAG, 202x.

[Int2x] Intersystems Resources. Retrieval augmented generation. https://www.intersystems.com/resources/retrieval-augmented-generation/, 202x.

[K2v2x] K2view Blog. What is retrieval-augmented generation. https://www.k2view.com/what-is-retrieval-augmented-generation, 202x.

[Kam2x] G. Kamradt. Llmtest_needleinahaystack, 202x. GitHub repository.

[Lan2x] Langchain Documentation. Build a retrieval augmented generation (rag) app: Part 1. https://python.langchain.com/docs/tutorials/rag/, 202x.

[Med2xa] Medium Article. Demystifying transformers tokenizers. https://medium.com/@weidagang/demystifying-transformers-tokenizers-961430c43ae6, 202x.

[Med2xb] Medium Article. Understanding rag evolution, components, implementation, and applications. https://medium.com/@sandyeep70/understanding-rag-evolution-components-implementation-and-applications-ecf72b778d15, 202x.

[Med2xc] Medium Article. Understanding the rag architecture model. https://medium.com/@hamipirzada/understanding-the-rag-architecture-model-a-deep-dive-into-modern-ai-c81208afa391, 202x.

[Pro2xa] ProArch Blog. All about the hallucination metric in large language models (llms). https://www.proarch.com/blog/all-about-the-hallucination-metric-in-large-language-models-llms/, 202x.

[Pro2xb] Prompting Guide. Research on rag. https://www.promptingguide.ai/research/rag, 202x.

[Res2x] Restack Article. Tokenization knowledge answer: Tokenize words c++ cat ai. https://www.restack.io/p/tokenization-knowledge-answer-tokenize-words-cpp-cat-ai, 202x.

[Sta2x] Stack Overflow. C++ tokenization. https://stackoverflow.com/questions/23445199/c-tokenization, 202x.

[Tim2x] Timeplus Blog. Building a rag-based question/answer system using ollama and timeplus. `https://www.timeplus.com/post/rag-based-question-answer-system`, 202x.

[Ton2x] Tonic AI Docs. Rag components summary. `https://docs.tonic.ai/validate/about-rag-metrics/tonic-validate-rag-components-summary`, 202x.

[Tow2x] Towards Data Science. The needle in a haystack test. `https://medium.com/towards-data-science/the-needle-in-a-haystack-test-a94974c1ad38`, 202x.

[Wil2x] WillowTree Apps Blog. Retrieval augmented generation. `https://www.willowtreeapps.com/craft/retrieval-augmented-generation`, 202x.