Assignment 4: Generation Probability Trees, Prompt Engineering and Agentic Systems

Deadline: Monday November 10, 2025 at 9:00pm
Late Penalty: There is a penalty-free grace period of one hour past the deadline. Any work that is submitted between 1 hour and 24 hours past the deadline will receive a 20% grade deduction. No other late work is accepted.

The goal of the fourth assignment is to gain familiarity with the the probabilistic nature of language generation with Transformers, and to gain skill in prompt engineering for generation and classification as well as using multiple LLMs as interacting agents in an *agentic* approach to creating systems. This will be helpful for an agentic approach to your course project.

This assignment must be done individually. The specific learning objectives in this assignment are:

1. Learn about the generation (decoding) process and viewing the generation probability tree, as well as the effect key generation-controlling hyperparameters.
2. Becoming literate in using the very best LLMs through API calls over the internet.
3. Become practiced in the prompt engineering methodology described in class and in this document - applying it to generation and classification, and specifically using chain-of-thought prompting
4. Building an agentic system for conversational generation (using rudimentary synthetic humans) and the evaluating the outcome using an evaluation LLM, prompted with an rubric.
5. Using those OpenAI-compatible APIs to interact with an LLM running locally on your own computer. (bonus)

What To Submit

You should hand in the following files to this assignment on Crowdmark:

- Answers to the written questions in this assignment. Be sure that you write the answers in the textbox, and don't include files instead. These questions are numbered in the form `Question X.Y.Z`, where X is the section of this assignment, Y is the subsection, and Z is the numbered element in the question.
- Code files as specified in individual questions.

1 Exploration of Text Generation Parameters and Probabilities (12 points)

Review the Huggingface documentation on performing auto-regressive text generation, which you encountered to a limited extent in Assignment 3, which can be found here. In particular take a copy of the code titled Multinomial Sampling.

Make sure you can run that code, which downloads a (medium-sized) GPT2 model and uses it to generate text given the specific input prompt.

The `model.generate` function (which has the same goal as the generate function you worked with in Assignment 3) has many parameters, including temperature and top_p as described in class. These parameters influence the sampling of the output probabilities which are used to select each word in turn in the auto-regressive generation loop. There are two other parameters to become familiar with which control how many tokens are generated: `max_length` and `stopping_criteria`. There are also parameters such `repetition_penalty` which penalize repeated words, and reduce their occurrence if the penalty is set above 1. There are many parameters of the `generate` function and you can read about them here.

Consider the following input sequence: "It is important for all countries to consider protection of human rights because". In the following steps you are asked to both generate subsequent words from that input context using that GPT2 model, and to explore the probabilities that are generated.

1. To get a sense of the influence of some of the generation parameters, explore at least 5 combinations of temperature and top_p to generate a maximum of 30 tokens using auto-regressive generation with the GPT2 model. Comment on how the generation differs across the range of parameters that you have selected. You must choose your own range, and you'll have to do some exploration to do that; you are free to explore other parameters if you wish. [5 points]

2. Modify the code to output the probabilities of the each word that is generated. You'll need to set these two generate parameters:`return_dict_in_generate=True` and, `output_scores=True`, and extract the probabilities that come in the returned dictionary one call at a time. Provide a table that shows these probabilities, similar to Assignment 3. Comment on the probabilities. [2 points]

3. Write new code that generates the probability tree (like the one drawn on the board in Lecture 6 and shown on page 6-5 of the notes in Lecture #6 Part 1) using the `treelib` package that you can find here. Generate the tree for the above sequence as input, providing the top 2 probabilities for each word position, as far as is practical to see, and submit that as part of the answer to this question. (You'll have to apply some common sense here to visualize the tree). Comment on the what you see in the tree. Is the tree affected by the top_p parameter or the temperature parameter? Why or why not? Submit your full code that runs the generation and builds and outputs the tree in the file `A4_1_3.py` [5 points]

2 Accessing OpenAI, Setting Limits, Learning the API (6 points)

For several parts of this assignment, you'll be using models from OpenAI - either through the usual chat interface, but mostly by direct programmatic API calls. If you do not have an OpenAI account already you should sign up for one here: https://auth.openai.com/create-account. You will need to put in your own credit card to use the API, and so the work in this assignment will incur expenses, but the costs per token are quite low (see here for prices). The next sections should cost well below $2 USD. That said, **you will be programmatically accessing the OpenAI API, which means you could accidentally put your code into a loop that could incur significant expenses.** To *prevent* that from happening, make sure you only load your account with a few dollars.

For some of your work in the following sections, you are required to use the OpenAI API to access the models, rather than the highly manual process of cutting and pasting to the chat interface. To learn how to do that, read https://platform.openai.com/docs/overview and then run the

code shown at the beginning of this link: https://platform.openai.com/docs/quickstart?context=python&language=python.

1. Run the three examples of getting an OpenAI model to analyze a picture, a website given by the URL, and a file of your own choosing. Give the examples you used to show the three modalities (image, URL and file), and the prompt/queries you used, which must be different from the ones shown in the documentation. [3 points]
2. Next, read the Core Concepts section of the OpenAI document, starting here: https://platform.openai.com/docs/guides/text, to the end, including the 'Migrate to Responses API' section.
3. Read the 'Agents' section of the documentation
4. Read the 'Tools' section of the documentation, including MCP.
5. Write a paragraph on how you might use Response, Agents and Tools in your project. Here we are looking for something insightful and thoughtful. [3 points]

3 Prompt Engineering Methodology

Recall the discussion in Lecture 7 about Prompt Engineering - the design of english-language statements to stimulate a large language model to perform a specific task. The methodology describe in Lecture 7 is reprised below. The context of the approach is either the generation of text or classification of text, working on an input example that you wish to either generate from, or to classify. Call that example a training example, and if it is for classification, then it would come with a correct label. If it is for generation then it won't necessarily come with a label, but it might come with a 'correct' generation.

Assume also that you've got 30 training examples, and divide them into 10 examples for 'training' as described below, and the remaining 20 as your hold-out test set. One important insight is that prompt engineering-based generation or classification still requires that there be a training set, and a hold-out test-set. The good news, though, is the number of examples needed in these sets appears to be *much* less than required in *either* in fine-tuning a pre-trained model, *or* training a model from scratch.

General Prompt Engineering Method

1. Write down your criterion for what makes the output acceptable, in English. This may involve looking up the definition of one or more words of the goal, or exploring other online resources so that *you* clearly understand it. This applies to both generation and classification.
2. Draft a Prompt that uses that criterion to direct the model to generate what you want. This 'generation' can be the classification of what the input is, or it could be the generation of some full text.
3. Using just **one** input example (which in classical machine learning training would be called a training example) run the prompt and the example to see how well it works, in the OpenAI normal chat interface, on the model.

4. Inspect the 'output.' If it is not correct in every way, evolve the prompt, using English, to make it work perfectly. This means changing the prompt to correct anything that is wrong in the output. However, *do not* use language that is specific to the input example, as that will not generalize. Notice that the concept of generalization is showing up here, but in a quite different way. As long as you don't 'cheat' - using words specific to an example, then a prompt that uses 'general' language should generalize! Also note that, in the world of prompt design, it will be helpful to cultivate an ability to write clearly, with good use of language and meaning of words.
5. Once you've made it work for one input example, make it work for a second example, using the same method - evolving the same prompt and correcting any issues seen on the second example. Make sure that you didn't break the first example in making the second example work. See the example used in Lecture 7 for some insights.
6. Then, try the prompt on five more training input examples all at once, and label the outputs as good/not good with respect to your criterion. Evolve the prompt to succeed on all five, but again, not breaking the previous 2.
7. Test. Run the prompt on your 20 example hold-out test set. Measure success rate yourself, i.e. with human labeling. Report your success rate.
8. If the success rate is below 100% you could choose to iterate once again, adjusting the prompt to correct the unsuccessful examples.

There are many suggestions online about how to evolve prompts, including the Medium article survey that was posted with Lecture 7 on Quercus (PromptEngineering_Medium-1.pdf).

4 Prompt Engineering for Generation of Soft, Non-Expert Therapeutic Statements (14 points)

In talk therapy counselling it is often important to make observations based on a patient's words that give insight to the patient. However, it is known that making direct and strong statements (e.g. "You are addicted to cigarettes") can be received by a patient as an accusation or unwanted label, which a patient will likely reject. This is made more problematic if the language used by the therapist implies that the therapist is superior or in a higher-up position of some kind - as an expert or because of their position "above" the patient, and so the words are correct simply because of that position. (For example, to say "I can see that you have many problems" is a very unpleasant and authoritative thing for a therapist to say).

Talk therapy is known to be far more effective observations/statements from the therapist are are expressed as possibilities, not facts. One approach is to convert statements to questions. Also, statements that suggest the patient knows the truth better than the therapist (it is the patient's life after all) are also known to lead to better outcomes.

In general, encouraging a patient to come to their own conclusion is known to be more successful. For example, rather than say "You're afraid of being judged by your family," a softened statement might become "I might be wrong, but is it possible that you're worried that your family will judge you?" In the latter statement the speaker is both ceding expertise to the patient, and more gently suggesting a possibility, rather than strongly stating a fact.

This softened statement is part of a broader therapeutic approach called Motivational Interviewing, which if you're interested, you can read about here: https://www.ncbi.nlm.nih.gov/books/NBK571068/.

Your task in this question is to engineer a prompt for GPT-5 to convert direct statements to what we will call *softened, non-expert* statements.

The dataset given in the file `DirectStatements.csv`, has a set of examples consisting of statements that are too direct, and are to be softened. These statements came from a variety of places in behaviour change counseling, generally relating to addictions.

Follow the steps below, which are modeled on the methodology given in Section 3.

1. Write, in your own words (not those above), a clear definition of what it means to convert a statement into a softened, non-expert version. Give your definition. [2 points]
2. Follow steps 2 through 4 of the methodology of Section 3 using the GPT-5 model using OpenAI's normal chat interface (not the API). This means you work only on the first training example (in row 1) in the dataset, until you are satisfied that the result is good. Report the prompt that you arrived at in step 4. Produce three different softened versions of the first example, and say for each why it meets your definition. You may need to explore different settings of the 'thinking' parameter or maybe just pushing the button more than to get different answers.[4 points]
3. Generate a result on the second item (row 2), and explain how it meets your definition. [1 point]
4. Give the result on the next five examples (rows 3-7), and any changes you make to the prompt to make them all succeed. [1 point]
5. Using the Open API that you read about in Section 2, (and *not the chat interface*) run the remaining 23 examples and determine, by hand, if they meet your criterion. Submit a csv file that contains three columns: the first input statement, the second column for the produced output, and the third column that gives a label that indicates if output meets the criterion (label 1) or not (label 0). Name that file `A4_4_4.csv`. Report your resulting success rate, and which result you think is the best, and which is the worst. Provide the full code that you used in a python file `A4_4_4.py`. [6 points]

5 Prompt Engineering for Classification of "Softness" (8 points)

In Lecture 7 we also discussed and illustrated a prompt that turned a large language model into a classifier. In this section you will create a classifier that determines if a statement has the softness feature described in Section 4.

1. Create a dataset that combines both the input statements from Section 4 and all of your outputs for all 30 examples, with labels. Assume that all of the inputs in the original file `DirectStatements.csv` would have a label 0, and use your assigned label from Part 4 of Section 4. Submit the full dataset in a file named `A4_5_1.csv` [1 point].

2. Using the method described in Section 3, and the combined data set you just created, evolve a prompt using 6 examples (3 negative, 3 positive). Show the prompt and give the success rate across those 6 examples. Try both with and without the method of *Chain of Thought Prompting*, as discussed in Lecture 7, and in the post from Medium and the paper https://arxiv.org/pdf/2201.11903.pdf. See if it makes a difference. [3 points]
3. Using the OpenAI API, run the remaining examples and give the success rate. Attempt to explain any incorrect results. [4 points]

6 An Agentic System to Simulate and Evaluate a Child-Parent Conversation (15 points)

As discussed in class, more complex AI systems are now being built using *Agentic* systems which consist of several instances of a large-language model all collaborating towards a specific goal. Each of these instances will have its own function, driven by its own prompt, and custom software connects the different instances. The software may modify prompts given to the instances.

In this section you are to build a four-instance system, using GPT-5 to create a conversation between a three-year old child who doesn't want to eat their dinner, and his two parents. The child, the first parent, and the second parent should all be separate instances of a GPT-5, with their own prompts.

A fourth instance, create a conversation evaluator (called, in the literature, "LLM-as-a-judge"), will produce a score from 1-5 of the quality of the conversation with respect to the criteria expressed here: https://kidshealth.org/en/parents/toddler-meals.html You will need to turn the expertise provided on this website into a *rubric* that says how well the conversation went, with a score of 5 being good. To be clear, you need to create a prompt that describes the criteria by which the full text of the conversation should be judged.

You will need to choose a personality for the child, and the parents, and cause the three models to engage with each other. You can code this in straight python, or, if you wish, make use of OpenAI's agentic framework (that you read about in Section 2) or some other agentic framework.

1. Create the API-based software to make all of three agents interact, giving each agent a view of the conversation so far. Design your first prompts for each of the three agents, and run the conversation so that each agent gets at least three turns, starting with the first parent trying to get the child to eat their dinner. Give the transcript of this first conversation, and then offer your own commentary on how realistic it is. [5 points]
2. Make two versions of the child, one where it is relatively easy to get along with, and quickly does try to eat the food offered. The second should be very difficult to convince. Once you're happy with the prompts for these cases, provide them in the file `ChildEasy.txt` and `ChildHard.txt`, along with the transcripts of the conversation `Easy.txt` and `Hard.txt`. Comment on how well the conversations went in both cases. [5 points]
3. Provide your rubric for your evaluator prompt in a file labelled `evaluator_rubric.txt`. Run your evaluator on the transcripts of the two full conversations above, and provide the output. Comment on the quality of the evaluation. [5 points]

7 Locally Hosted Models (6 Points)

It is now possible to run smaller models on your own computer, and to use those models with the identical APIs available from OpenAI and other providers.

1. Go to https://lmstudio.ai and download `lmstudio` to run on your own computer.
2. Using the search feature (click the search icon) download an open source model (based on the 'staff picks') that will fit and run on your computer. I would suggest, if it fits, to try the openai/gpt-oss-20b model. It can fit in a memory size of 16Gbytes, I believe, either in your CPU or GPU (which is the same if you have modern mac, but not windows).
3. Read the documentation for lmstudio, here: https://lmstudio.ai/docs/app, particularly the OpenAI compatible endpoints, described here: /urlhttps://lmstudio.ai/docs/developer/openai-compat. These allow you to use your code from this assignment directly on your own computer, without going out into some other entitites' model. This has big implications for creating privacy and control of this important kind of AI in the future.
4. A key question is: how good are these "open source" models? If they can do much of what the for-pay models do, then you can run them much more cheaply (perhaps) and in your direct control.
5. Run the code/examples you created in Section 4 part 5 using the open source model you selected. Provide its output. Comment on the quality comnpared to GPT-5. [2 points]
6. Run the code/examples you created in Section 5 part 2 using the open source model you selected. Provide its output. Comment on the quality comnpared to GPT-5. [2 points]
7. Run the code/examples you created in Section 6 using the open source model you selected. Provide its output. Comment on the quality comnpared to GPT-5. [2 points]

8 Generative AI Disclosure

After you have finished answering all the questions in Crowdmark for this assignment, describe the nature of your use of Generative AI in doing the work of this assignment. This question is required to be answered, and assignments will not be accepted without an answer.