Building Systems with the ChatGPT API

https://learn.deeplearning.ai/chatgpt-building-system/lesson/1/introduction

Introduction:

- Welcome to the course on building systems using the ChatGPT API.
- Previous course focused on prompting ChatGPT, but a system requires more than a single prompt.

- **Objective:**
  - Share best practices for building complex applications using a Large Language Model (LLM).
  - Example: Building an end-to-end customer service assistant system with multiple language model calls.
- **System Workflow Example:**
  - Illustrated with a user input: "Tell me about what TVs are for sale?"
    - Steps:
    - Evaluate input for problematic content (e.g., hateful speech).
    - Process input, identify the type of query (complaint, product info request).
    - If a product inquiry, retrieve relevant information about TVs.
    - Use a language model to generate a helpful response.
    - Check the output for potential issues (inaccuracy, inappropriateness).
- **Sequential Processing:**
  - Emphasizes the need for sequential processing of user input through multiple internal steps.
  - Many internal steps are invisible to end-users.
- **Long-Term Improvement:**
  - Building complex systems with LLMs involves continuous improvement.
  - Best practices for evaluating and enhancing the system over time.
- **Conclusion:**
  - This lecture aims to provide insights into building and improving systems with the ChatGPT API, using practical examples and best practices.

Lecture 1: Overview of Large Language Models (LLMs):

- Explains how LLMs work, including training, tokenizer, and chat format.
- Describes the training process using supervised learning with labeled data.

- **Training LLMs:**
  - Supervised learning core building block for training LLMs.
  - LLMs predict the next token in a sequence, not just the next word.
  - Two major types: "Base LLM" and "Instruction Tuned LLM."
- **Instruction Tuned LLM:**
  - Aims to follow instructions for specific outputs rather than general predictions.
  - Training process involves fine-tuning on a smaller set with human-rated feedback.
- **Tokenization and Challenges:**
  - LLMs predict next tokens, not individual letters.
  - Tokenization breakdown illustrated with examples like lollipop.
  - Trick to improve performance in specific tasks using tokenization.
- **Token Limits:**
  - Different LLMs have varying limits on input plus output tokens.
  - GPT-3.5 Turbo, a commonly used model, has a limit of around 4,000 tokens.
- **Using Messages Format:**
  - Introduces a method of specifying separate system, user, and assistant messages.
  - Demonstrates with examples using a helper function for prompts.
- **Secure API Key Handling:**
  - Advises against exposing API keys in plain text.
  - Recommends using a library like "dotenv" to load API keys securely.
- **Prompting Revolution:**
  - Contrasts traditional ML workflows with prompting-based ML.
  - Highlights the speed and efficiency of building applications using prompting.
  - Revolutionizing AI application development, especially for unstructured data.
- **Caveat for Structured Data:**
  - Acknowledges that the prompt-based approach may not be as effective for structured data applications.
- **Workflow Changes:**
  - Revolutionizes the speed at which AI components can be built.
  - Shortens the time frame for building text-based AI applications significantly.

Lecture 2:

- **Focus on Input Evaluation:**
  - Emphasis on tasks for evaluating inputs for quality and safety of the system.
- **Classifying Queries:**
  - Beneficial to classify the type of query first, especially for tasks with diverse instructions.
  - Use fixed categories and hard-code instructions relevant to each category.
- **Example: Customer Service Assistant:**
  - Illustrates building a customer service assistant by classifying queries.
  - Different secondary instructions based on the type of user query.
- **System Message and Delimiter:**
  - Uses a system message with a delimiter (#) to structure instructions.
  - Delimiter helps separate different sections of an instruction.
- **Structured Output:**
  - Requests classification output in JSON format with keys for primary and secondary categories.
  - Enables easy parsing into objects like dictionaries for subsequent steps.
- **Example User Messages:**
  - First example: User wants to delete their profile and data, categorized as account management.
  - Second example: User asks about flat-screen TVs, categorized appropriately.
- **Structured Output Benefits:**
  - Structured output allows for more specific instructions based on the categorized customer inquiry.

Lecture 3

- **Ensuring Responsible System Usage:**
  - Importance of checking user responsibility and preventing system abuse.
  - Introduction of strategies to achieve this.
- **Content Moderation with OpenAI's Moderation API:**
  - Overview of OpenAI's Moderation API for content compliance.
  - Reflects commitment to safe and responsible AI use.
  - Identifies and filters prohibited content in various categories (hate, self-harm, sexual violence).
  - Offers subcategories for precise moderation.
- **Moderation API Example:**
  - Demonstration using OpenAI Python package (**openai.Moderation.create**).
  - Example input flagged for violence with category scores.
  - Possibility of customizing policies based on category scores.
- **Prompt Injections:**
  - Definition: User attempts to manipulate AI system by overriding intended instructions.
  - Risks of unintended system usage and the need to detect/prevent prompt injections.
- **Strategies to Avoid Prompt Injections:**
  - **1. Delimiters and Clear Instructions:**
    - Using delimiters and clear system instructions to avoid prompt injections.
    - Example with a system message enforcing responses in Italian.
  - **2. Additional Prompt for Injection Detection:**
    - Asking the model to output Y or N to determine if a user is trying to inject conflicting or malicious instructions.
    - Example with system instruction to always respond in Italian.
- **Examples and Model Responses:**
  - Demonstrations of good and bad user messages.
  - Model's ability to classify user messages as prompt injections.

Lecture 4

- **Introduction:**
  - Focus on splitting complex tasks into simpler subtasks using multiple prompts.
  - Questioning the need for this approach when language models are adept at following complex instructions.
- **Analogies:**
  - Analogy 1: Cooking a complex meal all at once vs. in stages.
  - Analogy 2: Reading spaghetti code in one file vs. a modular program.
- **Benefits of Chaining Multiple Prompts:**
  - Breaks down task complexity for easier management and reduced errors.
  - Useful for tasks with potential ambiguity and complex dependencies.
  - Effective for workflows where maintaining the system state is essential.
- **Reducing Costs and Testing:**
  - Lower costs as longer prompts with more tokens are more expensive.
  - Easier testing of individual steps, identification of failures, and human intervention at specific steps.

- **Determining Complexity:**
  - Problem complexity arises when many instructions could apply to any given situation.
  - Developing intuition on when to use chaining prompts versus a single prompt.
- **Additional Benefits:**
  - Allows the model to use external tools at certain points in the workflow.
  - Examples include looking up information in a catalog, calling an API, or searching a knowledge base.
- **Example Workflow:**
  - Demonstrates using multiple prompts to answer a customer's question about products.
  - Breaks down steps, including looking up product information, into a series of subtasks.
- **Selective Loading of Product Descriptions:**
  - Discusses the decision to selectively load product information instead of including all descriptions in the prompt.
  - Reasons include potential confusion for the model, context limitations, and cost considerations.
  - Emphasizes language models as reasoning agents requiring relevant context for useful conclusions.
- **ChatGPT Plugins and Information Retrieval:**
  - Mentions the concept of ChatGPT plugins, where the model is informed about available tools.
  - Introduction to using text embeddings for efficient knowledge retrieval and fuzzy/semantic search.

Lecture 5

- **Introduction:**
  - Focus on checking outputs generated by the system.
  - Importance of quality, relevance, and safety in responses.
- **Moderation API for Outputs:**
  - Revisiting the moderation API, now in the context of checking outputs.
  - Example of using the moderation API to filter and moderate generated responses.
- **Adjusting Thresholds:**
  - Suggests adjusting moderation thresholds based on the target audience.
  - Importance of checking outputs, especially for chatbots serving sensitive audiences.
- **Model Self-Evaluation:**
  - Alternative approach: asking the model to rate the quality of its own output.
  - System message defining evaluation criteria and response format.
- **Example Scenarios:**
  - Example 1: Model evaluates a response related to customer service agent criteria.
  - Example 2: Model assesses a response to determine if it correctly uses retrieved information.
- **Advanced Model Considerations:**
  - Suggestion to use more advanced models like GPT-4 for better reasoning.
  - Experimenting with generating multiple responses and letting the model choose the best one.
- **Practical Considerations:**
  - Moderation API is good practice for checking outputs.
  - Caution about asking the model to evaluate its own output due to potential latency and cost issues.
  - Not commonly seen in production; may be unnecessary for most cases.

Lecture 6

- **End-to-End Customer Service Assistant:**
  - Integration of previous learnings to create an end-to-end example.
  - Steps involved in the process.
- **Processing User Message Function:**
  - Introduction of a function called "process_user_message."
  - Key steps: input moderation check, product extraction, product lookup, answer generation, and output moderation.
- **Example Interaction:**
  - Use of a sample user input about SmartX Pro phone, camera, and TVs.
  - Running through the steps of moderation, product extraction, lookup, response generation, and final moderation.
- **Code Explanation:**
  - Overview of the "process_user_message" function.
  - Handling flagged inputs and extracting product information.
  - Answering user questions and putting the response through moderation.
- **Chatbot UI Display:**
  - Introduction of a function to accumulate messages for a chatbot UI.
  - Displaying the chatbot UI and interacting with the customer service assistant.

- **Interactive Conversation:**
  - Asking about available TVs and inquiring about the cheapest and most expensive.
  - The assistant providing responses based on the processed information.
- **System Evaluation and Improvement:**
  - Encouragement to monitor the system's quality across various inputs.
  - Potential improvements in prompts, steps, retrieval methods, etc.

Lecture 7

- **Building and Evaluating an Application:**
  - Overview of building an application using a Language Model (LLM).
  - Emphasis on evaluating inputs, processing, and final output checking.
- **Evaluating LLM Outputs:**
  - Challenges in evaluating LLM outputs after system development.
  - Quick building of applications leads to gradual collection of test examples.
  - Distinction from traditional machine learning approaches due to rapid development.
- **Incremental Testing Process:**
  - Traditional supervised learning involves predefined test sets.
  - In LLM applications, incremental testing starts with a few examples and grows.
  - Tuning prompts with a small number of examples initially.
- **Fine-Tuning Prompts:**
  - Systematically adding tricky examples to development sets.
  - Manual evaluation becomes inconvenient with a growing set, leading to metric development.
  - If the system works well enough, stopping the evaluation process.
- **Larger Test Sets:**
  - Collecting a randomly sampled set of examples for further tuning.
  - Importance of larger test sets for fine-tuning in high-performance scenarios.
  - Use of hold-out test sets for unbiased, fair estimates of system performance.
- **Caveats in Evaluation:**
  - Importance of rigorous testing, especially in high-stakes applications.
  - For low-risk applications, stopping early in the evaluation process may be feasible.
- **Practical Example with Prompts:**
  - Use of prompts to extract relevant categories and products based on user input.
  - Evolution of prompts through examples, troubleshooting, and fine-tuning.
  - Introduction of automated testing for larger sets of examples.
- **Automated Testing Function:**
  - Demonstrating a function for automated testing on a set of examples.
  - Printing customer messages, ideal answers, and model responses.
  - Evaluation scores for correctness based on the ideal answers.
- **Iterative Workflow with Prompts:**
  - Iterative workflow in building applications using prompts.
  - Faster pace of iteration compared to traditional supervised learning.
  - Surprising effectiveness of evaluation based on a few hand-curated tricky examples.
- **Conclusion and Next Steps:**
  - Overview of the evaluation process and its effectiveness.
  - Acknowledgment of the difference in evaluating outputs with clear or ambiguous right answers.
  - Preview of the next video focusing on evaluating outputs in ambiguous settings.

Lecture 8

- **Evaluation of LLM Outputs:**
  - Previous video focused on evaluating LLM with clear right answers.
  - Introduces the challenge when LLM generates text without a single correct answer.
- **Rubric for Evaluation:**
  - Helper functions to obtain assistant responses and customer messages.
  - Introduction of rubric as guidelines for evaluating text output.
  - Evaluation based on factual content, ignoring style, grammar, or punctuation.
- **Rubric Evaluation Example:**
  - Specifying a rubric for evaluating the assistant response.
  - Rubric checks if the response is based on provided context, includes relevant information, and avoids disagreements.
  - Running the evaluation and interpreting the results.
- **Consideration of LLM Model Version:**

    - Note on using ChatGPT 3.5 Turbo for evaluation.
    - Suggestion to consider GPT-4 for a more robust evaluation, especially if sporadic.
- **Design Pattern: Rubric Evaluation:**
    - Emphasizes the design pattern of using rubrics to evaluate LLM outputs.
    - Rubrics allow for another LLM to assess the quality of the first LLM's output.
- **Comparison to Ideal Response:**
    - Introduction of another design pattern involving an ideal response.
    - Demonstrates using an expert-provided ideal response for comparison.
    - Suggests GPT-4 for more rigorous evaluations.
- **Use of Evaluation Framework:**
    - Reference to the OpenAI open source evals framework for rubric creation.
    - Encouragement for contributions to the framework for community collaboration.
- **Detailed Example: Ideal Response Comparison:**
    - Defining a prompt for comparing an LLM's output to an expert-written ideal response.
    - Rubric includes comparing factual content, ignoring style and punctuation differences.
    - Demonstrates how the LLM outputs a score based on the comparison.
- **Multiple Examples: Evaluation Outcomes:**
    - Evaluates two different assistant responses using the ideal response comparison.
    - One response gets an "A" as it is a subset and consistent with the expert answer.
    - Another response receives a "D" due to disagreement with the expert answer.
- **Key Takeaways: Design Patterns:**
    - Encourages two key design patterns for evaluation.
    - Rubrics enable LLM-to-LLM evaluation without an ideal response.
    - Using an expert-provided ideal response enhances the evaluation process.
- **Continuous Monitoring and Improvement:**
    - Importance of continuous monitoring and evaluation during development.
    - Tools provided for ongoing assessment and improvement of LLM system performance.