ShopTalk - AI-Powered Shopping Assistant

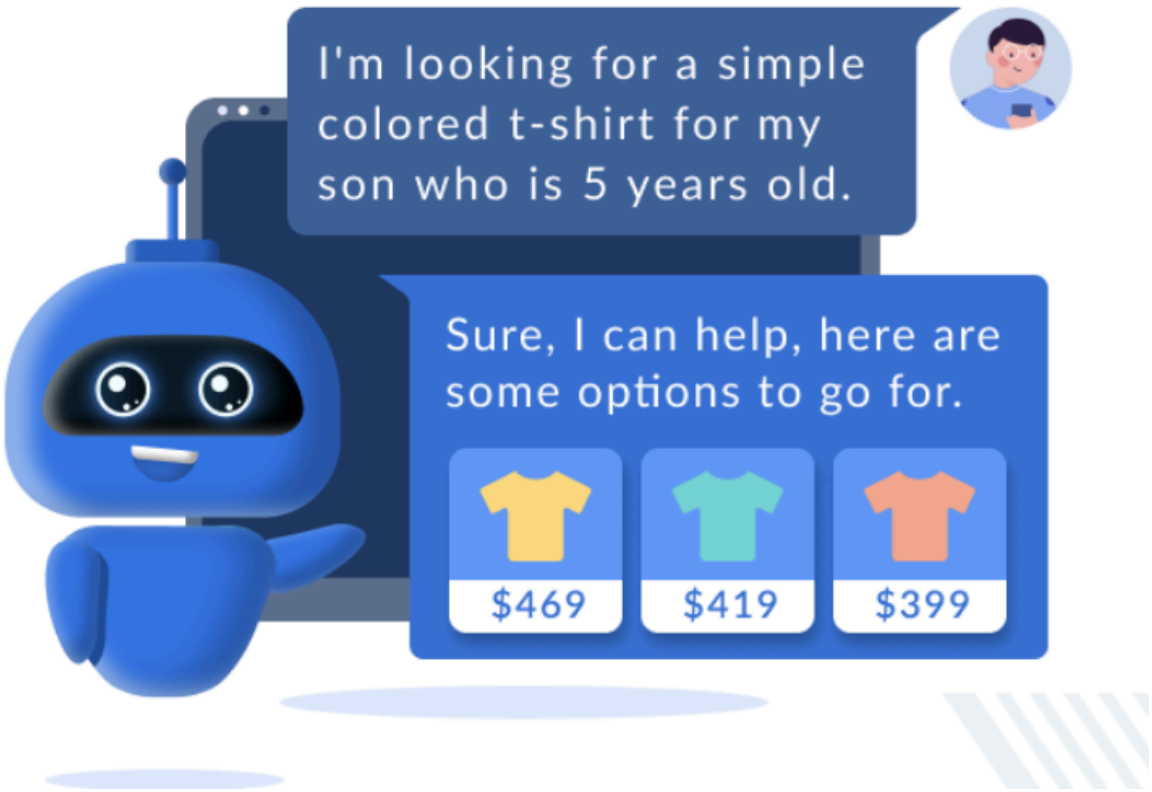
Motivation

Natural language interfaces are disrupting a lot of domains and providing more intuitive interaction of users with products and services in a conversational style. The shopping experience is going through a similar paradigm shift, where simple keywords based search is no longer an accepted norm, given it has severe limitations in terms of understanding the semantics of the knowledge base, as well as the user query. For example, it is very difficult to implement a keyword based search for queries like : “Show me a red shirt for men under 50 Dollars”. To provide such intelligent search, we can leverage modern-day large language models to build a Shopping Assistant for our users, leading to improved product discoverability, better search alignment to user preference, as well as improved overall NPS (Net Promoter Score) and business.

Problem Statement

We are supposed to build a smart assistant that, given a user query, can find the most relevant products from text and provide a natural language-based response.

The system's development focuses on multiple key aspects. First, it aims to achieve a sophisticated level of User Input Understanding, allowing the system to comprehend user inputs in natural language comprehensively, encompassing complex queries, synonyms, and variations. Additionally, the integration of the Retrieval Augmented Generation (RAG) model is essential to facilitate the efficient retrieval of relevant products based on user queries, ensuring both accuracy and speed in matching products to the query. Moreover, the incorporation of Natural Language Generation (NLG) capabilities is crucial to transforming identified products into coherent and human-like recommendations expressed in natural language. User Experience Optimization is another priority, emphasising the delivery of clear, concise, and contextually relevant recommendations, thereby minimising ambiguity and enhancing overall satisfaction. Finally, It would be great to come up with strategies to improve the latency of the bot.



Dataset

We recommend using Amazon Berkeley Objects (ABO) Dataset.
https://amazon-berkeley-objects.s3.amazonaws.com/index.html

Though you can use any other openly available datasets as well, the idea is to have product data with some sort of textual description of the products. The dataset should contain product images, since we need to caption the images and augment our existing text data with it.

The most useful data features for this project are product description, product keywords and Product Images.

Deliverables

(Optional) Fine Tuned Models

- Embedding generation pretrained models can be finetuned on the selected data, using loss functions like triplet loss in order to get better performing model customised for our dataset.
- Techniques like LORA and QLORA can be used to reduce the memory requirements during finetuning.
- The finetuned models can be compared with pre trained models using custom similarity metrics. One example :
    - Create similarity metrics based on normalised cosine distance of inter and intra product category.
    - Qualitatively analysing retrieval results.

Prototype

- A working Bot with basic UI (streamlit/ Gradio) as a rest endpoint on AWS
    - Bot, which takes user query as text input and outputs the required result.
    - Links/ ProductID or any other product identifier is also displayed

Core Functionality

- **Data Preprocessing:** Module to Preprocess the data in the required form. The preprocessed data should contain product details including category, brand, text description, image link, tags and any other relevant information.
- **Embedding Generation** : This module generates document embeddings based on existing pre-trained and finetuned models on our data.

- **Vector DB** : Set up open-source (Chroma/ Milvus) vector DBs and connect the Embedding Generation module to this DB.

- **Image Captioning**: Using image captioning models to generate text from product images and append it to product descriptions.

- **Large Language Models**: Explore different open source Large language models for performing the required generation tasks post retrieval.

- **RAG (Retrieval Augmented Generation):** Develop retrieval + Generation pipeline from scratch or using existing frameworks like Langchain. Deploy as a rest endpoint.

- **UI :** Develop a basic UI (streamlit/ Gradio, etc.) to connect to the RAG endpoint. Take textual input from the user and provide a relevant Generated result as well as a product identifier.

- **Feedback Loop (Stretch deliverable):** Design to utilise explicit/ implicit user feedback post-production launch

Submission Guidelines

- Submit all colab notebooks, along with Models created (Finetuned)
- Submit preprocessed Data along along with proper documentation for the process involved
- AWS code for RAG and inference along with documentation
- Clear Video recording of working bot.
- Ensure the "requirements.txt" file for installation of necessary libraries and downloadable datasets.
- Include any additional helpful instructions within the readme to enhance developers' understanding.
- Develop a separate document outlining the system architecture, algorithms, and implementation details for reference.
- Document initial results and experiences/comments
- Docker file and steps for deployment.

Model Evaluation and Testing

- Precision testing for retrieval task.
    - Perform precision analysis by identifying the percent of relevant results (True positives) from the *K* retrieved results. A test dataset needs to be created for this purpose containing query and possible positive products (matches).
- Qualitative testing for Generative results
- P95 and P99 results on Latency.
- A clear plan for further testing and improvement based on the evaluation results.

Optional Deliverables

- **Followup support**: Allow conversational history to be integrated to support follow-up queries.
- **Personalisation**: Allow personalised retrieval based on historical data at the user level.
- **Voice input**: Supporting Voice input along with the text.
- **Feedback Loop:** Implementation of feedback loop based on explicit(thumbs down) features.

Implementation Considerations

- **Cost:** Choose cost-effective technology options.
    - Colab/ Kaggle can be used for running experiments, fine tuning models, and getting code ready to be deployed on AWS.

- **Accuracy:** Try to get the best accuracies in open-source models
    - Compare accuracies of pre trained models on various benchmarks
    - Domain (Ecommerce) specific Models can be looked into

- **Latency**: A big factor in most Production systems.
    - The bottleneck will be LLM generation Latency
    - Multiple LLMs can be compared based on Latency with given Hardware.

Evaluation Criteria

1. **EDA and Data Preparation (Weightage: 15)**
   Demonstrates a profound understanding of the dataset through comprehensive Exploratory Data Analysis (EDA). Provides in-depth insights into key features and patterns. Language preprocessing is done, NLP preprocessing knowledge displayed. Data Preparation is meticulous, including thorough cleaning, transformation, and effective structuring of the data. Optimal preprocessing and feature engineering techniques are employed.

2. **Experimentation with Models (Weightage: 25)**
   In-depth fine-tuning of language model with clear explanation of modifications.
   Achieving high performance on language-related tasks (text generation/ embedding).
   Integration of Language and Image Models Well-Documented Fine-tuning Process and Inference. Comparisons done among multiple models. Code is well-organised, documented, and follows best practices. Clear explanation of fine-tuning process and inference steps.

3. **Deployment (Weightage: 25)**
   Demonstrates excellence by creating robust inference APIs, prioritising seamless integration and standardised model deployment. All the requirements are documented. Model is just loaded once during service up, not during inference. The Inference API uses the same data transformers used during training. Rest API is deployed, dockerization is required. Proper documentation created.

4. **E2E testing (Weightage : 15)**
   E2E testing done on a variety of test cases for Correctness and Latency measurement. Results Documented.

5. **UI/ UX (Weightage : 5)**
   Working UI with conversational history supported. Output and input properly rendered.

6. **Solution Documentation (Weightage: 15)**
   The entire implementation is well documented along with steps on how to set up operational tools, and rerun the entire pipeline or individual tasks. The system architecture of the pipeline is comprehensive, detailing what each step does. All libraries/ environments/ System configurations used should also be documented.

   Please note that there will also be a small project presentation by the students to the expert panel. The session will be planned sometime in mid-April 2024; more details will be shared later.

Future Directions

1. Finetuning Generative LLMs:

- Understand instruction based finetuning of LLMs along with optimisation techniques like LORA. Finetuning of LLMs helps us with updating any model with data from our domain which might not be present inside the pretrained model. Also instruction based finetuning can further help in learning novel tasks.
  Improving on Inference speed for open source LLMs is also something that can be looked into.

2. Image Input:

- Let's explore the scenario of an e-commerce company. Imagine an individual encounters a pair of shoes in person and captures a photograph of them. To enhance our system's capabilities, it becomes imperative to integrate support for image input. Our system should adeptly extract the underlying semantics from the photograph. This capability allows us to deliver highly relevant results to the user, aligning their visual input with our product offerings effectively.

3. Personalisation:

- How to use historical User Item interaction data, to show personalised results. It can either be part of the retrieval, or might come as a reranking problem post retrieval.

FAQs

What type of input and output does the Shopping assistant expect and provide?

The shopping assistant expect textual Search as input. It provides textual results with URL if available.

What are the recommended approaches and platforms for leveraging AI/ML libraries in this project? Are there any specific configurations to be aware of?

Creating a conda environment in AWS should be just fine.

What infrastructure is required for the assignment?

Colab :

1. Experimentation with Text Embedding models and Generation models

2. Image captioning experimentation and output creation (batch)

3. Preprocessing and general EDA of Product Data

4. Finetuning Text Embedding models

AWS :

G4dn.xlarge (Single GPU) ec2 instance.

1. Hosting inference pipeline.

Which front-end frameworks are recommended for the project?

Streamlit/ Gradio is a recommended front-end framework for web app development

What performance metrics should I focus on when evaluating the effectiveness of the Shopping assistant and NLP models?

Evaluate the Shopping assistant's and NLP models' retrieval precision, Qualitative e2e accuracy, efficiency, and user experience.

Can I use other frameworks or libraries not explicitly mentioned for specific tasks in the project?

While the project suggests specific libraries and frameworks, you are encouraged to explore other relevant tools based on your expertise. Ensure compatibility and consider the ease of integration with the existing project structure.

Core Milestones

1. Project Initiation
    a. Define project goals, objectives, and requirements.
    b. Set up the development environment, including necessary libraries and frameworks.
    c. Establish version control and collaborative development tools (e.g., Git and GitHub).
2. Research and Planning
    a. Conduct a literature review on Virtual assistant technologies, Natural Language and relevant LLM/ Image Captioning models.
    b. Define user stories and functionalities for the Shopping assistant.
3. Language Models experimentation

    a. Compare different Text embedding generation models. Measure effectiveness Qualitatively or quantitatively if possible.
    b. Compare different Vector DBs, pros and cons in performance.
    c. Finetune pretrained Embedding models.
4. RAG (Retrieval Augmented generation) creation
    a. Design RAG design, best practices and system flow.
    b. Implement RAG using open source frameworks like Langchain.
5. Inference API
    a. Design inference API for RAG supporting textual QA.
6. UI Development
    a. Implement a basic and user-friendly front end using frameworks like Streamlit/ Gradio.
    b. Ensure responsive design and ease of use for a seamless user experience.
    c. Conduct user acceptance testing for the UI.
7. Documentation
    a. Create comprehensive documentation covering system architecture, algorithms, and implementation details.
    b. Develop user manuals and guides for different functionalities.
    c. Ensure that all code is well-documented for future reference.
8. Final Testing and Evaluation
    a. Perform extensive testing of the complete Shopping assistant system.
    b. Evaluate performance metrics related to accuracy, efficiency, and user experience.
    c. Address any final issues or refinements based on testing outcomes.
9. Deployment and Presentation
    a. Prepare for project deployment, ensuring all components are ready for use.
    b. Create a compelling and informative presentation showcasing the project's features, functionalities, and applications.
10. Project Submission and Reflection
    a. Compile all project deliverables, including code, documentation, and presentation materials.
    b. Submit the final project for evaluation.
    c. Reflect on the project journey, lessons learned, and potential areas for future enhancements.
