

Technical Test

This technical test consists of two parts. You only have to complete one of the two. Depending on your expertise and experience you can choose to implement a purely NLP task or a CV/NLP Multi-Modal task.

General Requirements

Successful implementation will include:

- All the necessary steps and requirements to run and reproduce results
- Clear code with documentation
- A text document outlining the implementation process and time taken for each step
- SOTA concepts (feel free to cite relevant papers/sources)

Dataset

The Google Drive link contains an [articles.csv](#) with a list of Women's products. Each article/item has a free text title and description and some other categorical information. Most of the articles also have associated product images in the [images](#) directory. You are free to transform the data points or use additional datasets to support your ML methods.

Technical Test 1

NLP Task (Option A)

A conversational agent (using Python code) that can leverage multiple sources of product information to make informed recommendations via a chat interface. The agent should take as input customer messages. The output should be text with clarification questions or relevant product recommendations (include the article id). Focus on complex conversation scenarios with multiple messages.

Requirements:

Only recommend relevant products based on the customer conversation

The agent should ask clarification questions to narrow down the possible results (narrow down the types of products, colours etc.)

We are looking for creative solutions! A non-exhaustive list of ideas/examples (we encourage you to implement some creative solutions, also useful for discussions about further improvements):

Cite the information the agent used to answer a question (e.g., print the external sources used after the agent's response. Sources could be text/product attributes/image etc.)

Feature engineering (e.g.: incorporate multiple/different product properties into the product search)

Automated performance evaluation of the agent

Automated creation and optimisation of prompts

Use external sources of information by searching the web

Use some guardrail technique to prevent the agent from answering out-of-scope questions (e.g., "What car should I buy?", while the recommendation system is built for garments)

Use-case 1 - Product Advice

Customer (text message)

Do you have the Howie shorts in stock in blue?

Agent

Technical Test 2

The Howie shorts are a great choice!

We have a dark turquoise colour in stock.

****provide Howie Shorts in dark turquoise product id****

Use-case 2 - Guiding the customer through conversation

Customer

Hey, I am interested in buying some clothes

Agent

Hello, I'd be happy to assist you in finding the right products. Could you please provide some more information?

Is there a special event or activity you want to find an outfit for? Customer

I just want a new outfit for this weekend

you

We have a variety of clothing in stock, shall we start by getting you the right trousers, top and jacket combination?

Customer

That sounds good, although it's still quite warm, so shorts and a blouse will be enough.

you

We have a variety of patterns and colours in our collection of shorts. Do you prefer a solid colour, denim, striped, all-over pattern? Do you have any similar preferences for a blouse? For example, a lighter-coloured blouse could work well with denim shorts. Customer

The denim shorts sound good, maybe with a white blouse?

you

Great choice!

Here is the denim shorts and white blouse combination for your weekend. **provide a list of products ids**

Technical Test 3

Submission

Submit all project files necessary to run your project in a zip file or Github repository.

Include requirements.txt for Python dependencies.


If you train any models, include model weights and supporting files. Include all the datasets you've used/changed.

Datasets

Images dataset



Articles csv

 digital genius articles dataset

The name of this product is {prod_name}. It is a {product_group_name} , more specifically it is a {colour_group_name} {garment_group_name}, {product_type_name}. The graphical appearance of the product is {graphical_appearance_name}

article_id	prod_name	product_type_name	product_group_name	graphical_appearance_name	colour_group_name	garment_group_name	detail_desc
------------	-----------	-------------------	--------------------	---------------------------	-------------------	--------------------	-------------

				e			
0695255001	Siv t-shirt	T-shirt	Garment Upper body	All over pattern	Dark Blue	Jersey Fancy	Short-sleeved top in soft viscose jersey with a unique nursing feature. The design includes a double layer at the top to help retain warmth while allowing easier nursing access.
0821115007	RICHIE SKIRT	Skirt	Garment Lower body	Check	Pink	Skirts	Short, pleated skirt in woven fabric with a high waist and concealed zip and press-stud at one side. Unlined.
0553238008	THORN LS TEE	Sweater	Garment Upper body	Solid	White	Jersey Basic	Wide, long-sleeved top in soft cotton jersey with an open chest pocket, ribbing around the neckline and short slits in the sides. Loose fit.
0627147001	Bling Me Up Push	Bikini top	Swimwear	Lace	Dark Red	Swimwear	Fully lined bikini top with hole-patterned, underwired, moulded, padded cups that lift and shape, and ties at the back of the neck.
0794520001	Plus seam at back 1p tights	Underwear Tights	Socks & Tights	Solid	Black	Socks and Tights	Tights with a seam down the back of the legs. 30 denier.

0697564030	KELLY SHIRT S.0	Shirt	Garment Upper body	Stripe	Blue	Blouses	Shirt in airy cotton with a collar, buttons down the front, long sleeves with buttoned cuffs, and a rounded hem.
------------	--------------------	-------	-----------------------	--------	------	---------	--

Notebooks

- [redis-langchain-ecommerce-chatbot.ipynb](#)
- [Create an E-commerce Chat Application to Search Images And Text Using LangChain, Pinecone And Hybrid Search | by Plaban Nayak | Sep. 2023 | AI Planet](#)
- [Building AI-powered data-driven applications using pgvector, LangChain and LLMs](#)
- [Embracing AI-Powered Applications: A Developer's Journey with LangChain | Obytes](#)
- [Simple LangChain Code in Gradio](#)
- [langchain-claude-chatbot/conversation_chain.py at main](#)
- https://github.com/rahulapiit/ChatBot/blob/main/chatbot_demo.py
- https://github.com/tomasonjo/blogs/blob/master/neo4jdocs/neo4j_support_bot.ipynb?source=post_page-----63c4761193e7-----
- <https://github.com/RedisVentures/redis-langchain-chatbot/blob/main/redis-langchain-ecommerce-chatbot.ipynb>
-

Implementation

Research

This section covers all the areas i looked into during my research for this project. In total I probably spent an hour and a half on research. Some parts of this section have been taken from other research docs I created that were relevant to this coding challenge.

Model Architectures

Encoder-decoder or seq2seq

Encoder-decoder models (also called *sequence-to-sequence models*) use both parts of the Transformer architecture.. Sequence-to-sequence models are best suited for tasks revolving around generating new sentences depending on a given input, such as summarization, translation, or generative question answering.

Challenges

Overfitting: Seq2Seq models can overfit the training data if they are not properly regularized, which can lead to poor performance on new data. Handling Rare Words: Seq2Seq models can have difficulty handling rare words that are not present in the training data.

Models

- [Models - Hugging Face](#)



Encoders

Bert is an example of an encoder only model.

- Bi-directional: context from the left, and the right
- Good at extracting meaningful information
- Sequence classification, question answering, masked language modeling
- NLU: Natural Language Understanding
- Example of encoders: BERT, RoBERTa, ALBERT

Decoders

Gpt 2 is an example of a decoder only model

- Unidirectional: access to their left (or right!) context
- Great at causal tasks; generating sequences
- NLG: Natural Language generation
- Example of decoders: GPT-2, GPT Neo

Good for text generation and chat

Right now big LLMs are being used for prompting (In-context learning (ICL), Chain of thoughts) all of these tasks are based on text generation which have generally been seen to be performed well by decoder only models (GPT, GLM, Palm).

Autoencoder

One type of large language model is the autoencoder-based model, which works by encoding input text into a lower-dimensional representation and then generating new text based on that representation. This type of model is especially good for tasks like summarizing text or generating content.

LLM's for recommendation

- [LLM-Rec](#)
- [\[2305.19860\] A Survey on Large Language Models for Recommendation](#)
- [Augmenting recommendation systems with LLMs — The TensorFlow Blog](#)
- [Enhancing Recommendation Systems with Large Language Models | by Valentina Alto | Microsoft Azure | Medium](#)
- [Build a product recommendation chatbot for a Shopify store using OpenAI and Gadget](#)
 - In this tutorial, you will build a product recommendation chatbot for a Shopify store using OpenAI's API and Gadget. The chatbot will utilize OpenAI's text embedding API to generate vector embeddings for product descriptions, which will then be stored in your Gadget database. These embeddings will help to identify the products that best match a shopper's chat message.
- [Build an E-commerce Chatbot With Redis, LangChain, and OpenAI](#)
 - The ConversationalRetrievalChain that forms the chatbot operates in three phases:
 -
 - Question creation evaluates the input question and uses the OpenAI GPT model to combine it with knowledge from previous conversational interactions (if

any).

- Retrieval searches Redis for the best available products, given the items in which the shopper expressed interest.
- Question answering gets the product results from the vector search query and uses the OpenAI GPT model to help the shopper navigate the options.

○

●

- https://www.reddit.com/r/LangChain/comments/144q7ob/new_to_langchain_how_to_build_ecommerce_chatbot/

- 1.) write a prompt that takes a user's message and asks gpt to extract anything that sounds like a product name from it. You could also have this prompt return variations of the product name that would help fuzz it for searching (you could also have this be a separate prompt)

○

- We 2.) take those terms and use your existing search api to get results

○

- 3.) take the top few results and pass them in as context to a new prompt whose objective is to answer the user's query with the provided context of your products

○

LLM Evaluation

- [\[2306.05685\] Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena](#)
- [Best Practices for LLM Evaluation of RAG Applications | Databricks Blog](#)
- <https://github.com/explodinggradients/ragas>
-

RAG challenges

RAG is very difficult to do right. I am experimenting with various RAG projects from [1]. The main problems are:

- Chunking can interfere with context boundaries
- Content vectors can differ vastly from question vectors, for this you have to use hypothetical embeddings (they generate artificial questions and store them)
- Instead of saving just one embedding per text-chunk you should store various (text chunk, hypothetical embedding questions, meta data)
- RAG will miserably fail with requests like "summarize the whole document"
- to my knowledge, openAI embeddings aren't performing well, use a embedding that is optimized for question answering or information retrieval and supports multi language. SOTA textual embedding models can be found on the MTEB Leaderboard [2]. Also look into instructorEmbeddings
- the LLM used for the Q&A using your context should be fine-tuned for this task. There are several open (source?) LLMs based on openllama and others, that are fine tuned for information retrieval. They hallucinate less and are sticking to the context given.

1 <https://github.com/underlines/awesome-marketing-datascience/...>

2 <https://github.com/embeddings-benchmark/mteb>

Hypothetical embeddings

The inverse idea of Hypothetical Embeddings, HyDE [1] "HyDE is an embedding technique that takes queries, generates a hypothetical answer, and then embeds that generated document and uses that as the final example."

BriefGPT [2] is implementing this and it uses the following prompt at ingestion-time:

"Given the user's question, please generate a response that mimics the exact format in which the relevant information would appear within a document, even if the information does not exist. The response should not offer explanations, context, or commentary, but should emulate the precise structure in which the answer would be found in a hypothetical document. Factuality is not important, the priority is the hypothetical structure of the excerpt. Use made-up facts to emulate the structure. For example, if the user question is 'who are the authors?', the response should be something like 'Authors: John Smith, Jane Doe, and Bob Jones' The user's question is:"

1 <https://python.langchain.com/docs/modules/chains/additional/...>

2 <https://github.com/e-johnstonn/BriefGPT>

Semantic parsing (an old school methodology)

Extracting structured representations from unstructured, natural language text. We could combine an LLM with this classical methodology

- [Semantic Parsing for Task Oriented Dialog using Hierarchical Representations - ACL Anthology](#)
- [A Survey on Semantic Parsing](#)
 - semantic parsing, which is framed as a mapping from natural language utterances to meaning representations.
 - Semantic parsers map natural language (NL) utterances into a semantic representation. These representations are often (interchangeably) referred to as logical forms, meaning representations (MR) or programs
 - Wong and Mooney [2006] employ statistical machine translation (MT) techniques for the task of semantic parsing by arguing that a parsing model can also be viewed as a syntax-based translation model. They achieve good performance and develop a model that is more robust to word order
 - Learning from user feedback: Iyer et al. [2017] present an approach to improve a neural semantic parser based on user feedback wherein a rudimentary parser is used to bootstrap, followed by incorporating binary user feedback to improve the parse.
- [Weakly Supervised Training of Semantic Parsers](#)
 - able to train a semantic parser without needing any sentence-level annotations. Instead, they combine readily available syntactic and semantic knowledge to construct a versatile semantic parser that can be applied to any knowledge base

Example Query	Logical Form
capital of Russia	$\lambda x. \text{CITYCAPITALOFCOUNTRY}(x, \text{RUSSIA})$
wife of Abraham	$\lambda x. \text{HASHUSBAND}(x, \text{ABRAHAM})$
vocalist from London, England	$\lambda x. \text{MUSICIAN}(x) \wedge$ $\text{PERSONBORNIN}(x, \text{LONDON}) \wedge$ $\text{CITYINCOUNTRY}(\text{LONDON}, \text{ENGLAND})$
home of ConocoPhillips in Canada	$\lambda x. \text{HEADQUARTERS}(\text{CONOCOPHILLIPS}, x)$ $\wedge \text{CITYINCOUNTRY}(x, \text{CANADA})$

-
- <https://aws.amazon.com/datasets/freebase-data-dump/>
- Solving the long tail problem
 - <https://cobusgreyling.medium.com/solving-for-the-long-tail-of-intent-distribution-24daa372fc>
 - Create a custom loss function that takes into account the data imbalance. You may want to use a weighted loss function to give more importance to underrepresented intents and slots. Here's a basic outline of how you can define such a loss function:
 - **Long Tails:** The term "long tails" here refers to a situation where there are some intents that occur infrequently or have very few examples in the training dataset. In other words, there's a significant imbalance in the distribution of intents. Some intents are very common, while others are rare.
 - **Cardinality of Slots:** In the context of NLP, "slots" are specific pieces of information that need to be extracted from a user's input. For example, if the intent is "book a flight," slots might include "departure city," "destination city," "departure date," and so on. The "cardinality" of slots refers to the number of different slot values or categories.
- [Toward Code Generation: A Survey and Lessons from Semantic Parsing](#)
- [Low-Resource Domain Adaptation for Compositional Task-Oriented Semantic Parsing - ACL Anthology](#)
 - In this work, we study the low-resource domain scaling problem for task-oriented semantic parsing. In particular, we focus on the 25 SPIS setting to investigate whether a model can effectively adapt to new domains with a very limited amount of training data.
 - Task-Oriented Semantic Parsing has attracted attention from the research community since 1990s with the advent of the ATIS dataset (Price, 1990). Traditionally, the task is formulated as a joint text classification (intent prediction) and sequence tagging (slot filling) problem
 - In total, TOPv2 has 8 domains (alarm, event, messaging, music, navigation, reminder, timer, weather) and 180k samples randomly split into train, eval, and test sets for each domain. Please refer to the paper for more data statistics.
 - model training for low resource domain adaptation consists of two stages: source training and fine-tuning (or target training), where the model (initialized with pre-trained

representations) is first trained on the source domains and then fine-tuned on each low-resource target domain

- we propose to replace source training with optimization-based meta-learning (Finn et al., 2017) to improve generalization. Instead of directly optimizing towards source domain accuracy, meta-learning, when trained on the source domains, looks for a good initialization θ_0 that can easily be adapted to new tasks (domains) with minimal finetuning.
- Note: As TOPv2 data is provided on a per-domain basis, the UNSUPPORTED utterances in the original TOP dataset were removed as they could not be mapped to any domain.

Domain	Canonical semantic parse for the example utterance (see §5.2)
alarm	[IN:CREATE_ALARM [SL:DATE_TIME for noon tomorrow]]
event	[IN:GET_EVENT [SL:DATE_TIME tonight] [SL:LOCATION san francisco]]
messaging	[IN:SEND_MESSAGE [SL:CONTENT_EXACT yes] [SL:RECIPIENT bill] [SL:RECIPIENT mindy]]
music	[IN:REPLAY_MUSIC [SL:MUSIC_TYPE album]]
navigation	[IN:GET_INFO_TRAFFIC [SL:DESTINATION [IN:GET_LOCATION_HOME]]]
reminder	[IN:DELETE_REMINDER [SL:DATE_TIME this monday] [SL:TODO attend conference]]
timer	[IN:CREATE_TIMER [SL:DATE_TIME for 2 hours] [SL:METHOD_TIMER timer]]
weather	[IN:GET_WEATHER [SL:WEATHER_ATTRIBUTE cold]]

-
- Data Preprocessing We first perform standard preprocessing such as lower-casing and tokenization. For models initialized with pre-trained language representations, BPE (Sennrich et al., 2016) tokenization is done to match that used by the pretrained model. We do not tokenize ontology tokens (intents and slot labels) into BPE, but instead treat them as atomic tokens which are appended to the BPE vocabulary
- We then perform two additional preprocessing (canonicalization) steps, consistent across all models. First of all, note that certain utterance tokens do not contribute to the semantics of the query. For instance, in Figure 1, the phrase Driving directions to under IN:GET_DIRECTIONS and the under IN:GET_EVENT can be omitted as their semantics are already captured by the intent labels. Therefore, we only retain utterance tokens under leaf slots (Eagles and game in Figure 1) while removing all others. Furthermore, we sort the children of each node in the semantic parse tree in alphabetical order of the label, since the order of the children does not alter its semantic meaning. In the case of Figure 1, SL:NAME_EVENT and SL:CAT_EVENT will be reordered. We call the final semantic parse after these two preprocessing steps the canonical form, which will be used in all experiments.
- Adam is again used for optimization, with a learning rate of 10^{-4} for source training and 5×10^{-5} for finetuning. In addition, the inverse square-root learning rate schedule is employed with a warmup period of 4000 updates for source training, and 2000 for fine-tuning.
- For meta-learning, we select $k = 5$ and a batch size of 32 for Reptile, with both inner (η) and outer (α) learning rates being 5×10^{-5}
- Meta-Learning (Lake et al., 2015), or learning to learn, aims to learn a model that can quickly adapt to new tasks with a small amount of training data. In particular, Finn et al. (2017) propose MAML, an optimization-based meta-learning method, which learns a good parameter initialization suitable for faster adaptation to new tasks. As MAML requires to compute second derivatives, which are computation and memory intensive, there have

been studies to use either first-order approximation such as firstorder MAML and Reptile (Nichol et al., 2018), or implicit differentiation (Rajeswaran et al., 2019)

- [Compositional Task-Oriented Parsing as Abstractive Question Answering](#)

- <https://github.com/amazon-science/semantic-parsing-as-abstractive-qa>
- Task-oriented parsing (TOP) aims to convert natural language into machine-readable representations of specific tasks, such as setting an alarm
- We now present a general method for reducing compositional TOP to abstractive QA. Given an utterance, our goal is to recover its semantic parse tree by asking questions and parsing the answers returned by a QA model.

Multi-turn QA:

Q: A user may intend to get directions, get distance, get estimated arrival time, get estimated departure time, get estimated duration, get road condition information, get route's information, get traffic information, get location, make unsupported navigation, or update directions. A user said, "Look up directions to the nearest parking near S Beritania Street." What did the user intend to do?
A: get directions
Q: The slots for get directions may be locations, arrival datetimes, road conditions to avoid, waypoints, amounts, paths, sources, travel methods, road conditions, waypoints to avoid, departure datetimes, paths to avoid, obstructions to avoid, and destinations. A user said "Look up directions to the nearest parking near S Beritania Street." *The user's intent is to get directions.* What are the slots?
A: destination
Q: A user said "Look up directions to the nearest parking near S Beritania Street." *The user's intent is to get directions, and the slot is destination.* What is the destination?
A: the nearest parking near S Beritania Street
Q: The nested intent in destination may be get school's location, get home's location, get location, get event, and get workplace's location. A user said "Look up directions to the nearest parking near S Beritania Street." *The user's*

- <https://arxiv.org/pdf/2207.10643.pdf>

- we release STOP, the largest SLU dataset for end-to-end semantic parsing, provide initial baselines using both an end-to-end as well as a cascaded system, and show the importance of in-domain ASR training for cascaded systems.
- Traditional assistant systems utilize a cascaded approach: an Automatic Speech Recognition (ASR) system transcribes the audio to text, followed by a separate Natural Language Understanding (NLU) system which predicts the user's intentions from the decoded text. Although cascade systems enable using audio-only and textonly training resources, they are prone to error propagation when the ASR systems fail to generate the correct transcript. Further, an NLU model fed with decoded text cannot leverage paralinguistic information in the input, e.g., pausing, intonation, and stress variations.
- The STOP dataset builds upon the TOPv2 dataset [13] which provides text-only inputs and target semantic parse trees. For every utterance-semantic parse pair, we collect an audio recording of the input utterance by requesting workers through Amazon's Mechanical Turk to record themselves.

Table 1. Gender Statistics

Female	Male	Non-binary	Unanswered
44.55%	54.62%	0.79%	0.04%

Table 2. Native English Statistics

Native	Non-native	Prefer not to say
95.18%	4.09%	0.73%

Table 3. Spoken dataset comparison

	FSC	SNIPS	SLURP	STOP
Speakers	97	67	177	885
Audio Files	30043	5886	72277	236477
Duration[hrs]	19	5.5	58	218

- the demographics of the dataset is important
- [GitHub - SALT-NLP/DAMP: Baselines for TOP V2, MTOP, Hinglish TOP and CSTOP](#)
- <https://github.com/phuongnm94/PhraseTransformer>

Solution

Coding

Dataset processing

An essential part of this task, I had to pre process the dataset and transform the data into a format better suited to the coding challenge. This involved creating extra dataset columns that would then be used in the retrieval process. It probably took me 40 minutes to load the dataset, explore it, then decide on any augmentations I would make.

Prompt creation

My solution relies upon using the chat completion version of gpt -3.5, which requires prompting. In total i used 6 different prompts. It took me 30 minutes in total to create the prompts and optimise them.

Chat pipeline

This solution requires setting up the creation of a chat pipeline, complete with if/then statement to direct the user to the right chat response. This took me roughly 40 minutes to complete and optimise this pipeline. This includes light testing of the full pipeline.

Creative additions that have been implemented

- emotional _prompting (5 minutes) - [\[2307.11760\] Large Language Models Understand and Can be Enhanced by Emotional Stimuli](#)
- Tone prompting (5 minutes) - speak like donatella versace
- Prevent misuse and prompt injection (5 minutes) : [\[2306.05499\] Prompt Injection attack against LLM-integrated Applications](#)
- Cite the information the agent used to answer a question (5 minutes)
 - Cite the products that were returned in the query
 - Get the agent to choose the best one and explain why they made the choice
- Feature engineering ((5 minutes)
 - Turn product name, garment type, product category in to a natural language description
- Used some guardrail technique to prevent the agent from answering out-of-scope questions (5 minutes)

Last minute code clean up

After sending in my draft submission i had an opportunity to clean the code up somewhat. This included added more comments and fixing bugs in the chat pipeline. This took me an additional 30 minutes.

After my second submission I then made some further changes including allowing the user to interrupt the chat exchange using keyboard interrupt, and restricting the number of questions the agent asks before searching for a relevant product. This took me an additional 30 minutes.

Running the solution

To run the solution notebook provided, ensure that the articles.csv dataset is in the same directory as the notebook. Also ensure that the open ai key has been set. You will also need to make sure you have installed the requisite python packages. This can be achieved by running `'pip install -r requirements.txt'`.

Here are the pip installs that are required:

- `%pip install pandas`
- `%pip install -U sentence-transformers`
- `%pip install openai`
- `%pip install langchain==0.0.148`
- `%pip install tiktoken`
- `%pip install faiss`
- `%pip install numpy`
- `%pip install openai`
- `%pip install faiss-cpu`

After installing the requisite packages ensure you have jupyter notebook installed. Once installed kick off jupyter notebook from the project directory from the command line: `jupyter notebook`

Once all these steps have been completed, you can run the notebook by clicking 'run all'

Limitations

Restricted conversation flow

Due to time limitations and the way I have designed the chatbot pipeline, the questions that are asked of the user to elicit information about the product that will suit them are quite brittle and often leave little room for exploration.

Exceptions

The chatbot pipeline will sometimes run into situations where the open ai server times out, or is unavailable. Sometimes this will require restarting the kernel.

Speed

The time it takes for the chatbot to respond is quite high. This makes for a worse user experience

An example of a successful run

```
Hi! What are you looking for today?im looking for a dres
What is the occasion you're shopping for?a party
What is your preferred dress style for parties?sophisticated
```

```
Customer: I'm looking for a dress.
```

```
You: Hi! What is the occasion you're shopping for?
```

```
Customer: It's for a party.
```

```
You: Fantastic! What is your preferred dress style for parties?
```

```
Customer: I like sophisticated styles.
```

Based on the customer's preference for a sophisticated party dress, I would recommend the "Stunning dress" (article_id: 867357001) from our collection. This dress is a black, calf-length dress in jersey crêpe with a square neckline and long puff sleeves. It features gathers at one side to create a draped effect and a wrapover skirt with a slit at the front. The dress is lined in the front, making it both elegant and comfortable. The solid graphical appearance adds a touch of timeless sophistication to this stunning piece.

Additionally, I would also recommend the "Briony dress" (article_id: 863946001) as an alternative option. It is a black, calf-length dress in a cotton weave with broderie anglaise on the front of the bodice and down the sleeves. The round neckline, short puff sleeves with smocking, and narrow frill trim at the hems add an exquisite touch to this dress. The gathered tiers on the skirt provide added width and a stylish silhouette. This dress is unlined, making it lightweight and perfect for a sophisticated party.

Both dresses embody elegance and sophistication, reflecting your preferred dress style for parties. I hope you enjoy exploring these options and find the perfect dress for your special occasion!

Here are the products i used to make this recommendation:

```
[Document(page_content='The name of this product is Stunning dress. It is a Garment Full body , more specifically it is a Black Jersey Fancy, Dress. The graphical appearance of the product is Solid \n More details: \nCalf-length dress in jersey crêpe with a square neckline and long puff sleeves. Gathers at one side to create a draped effect, and a wrapover skirt with a slit
```