Prompt Engineering for Trip Itinerary Planning to LLMs through In-context Learning

Satya Mythili Nuthalapati
M.Sc, Computer Science
University of Florida
Gainesville, Florida
snuthalapati@ufl.edu

Vanshika Mehrotra
M.Sc, Computer Science
University of Florida
Gainesville, Florida
vmehrotra@ufl.edu

***Abstract*—In an age of information overload, travel planning can be overwhelming. Tourists yearn for personalized itineraries that maximize entertainment, but traditional recommender systems often fall short, struggling to handle the complex interplay of user preferences, real-time data, and dynamic contexts. This research presents a novel approach that leverages the power of large language models (LLMs) to transform travel planning into a truly personalized and context-aware experience.This amalgamation enables LLMs to function as intelligent travel assistants capable of explaining recommendations, fostering user trust and transparency.**

Prior conversational models produce repetitive generic outputs failing to meet diverse user needs. They rely solely on text and lack geographic context critical for trip planning. Our approach increases response diversity, incorporates multi-modal geographic data for deeper spatial understanding, uses iterative reinforcement learning to enable personalization and adaptation, increases diversity of responses. Furthermore, the system continuously learns, improves, adapts to user feedback and navigates through complex travel scenarios through curriculum learning. Automatic NLG metrics and human ratings show generated itineraries are more accurate, comprehensive, feasible and personalized compared to current state-of-the-art systems. This paradigm shift aims to empower users with the capability to craft personalized dream itineraries effortlessly and confidently, redefining the landscape of travel planning.

Keywords—Large Language Models, Travel Planning, Personalized Experiences, Prompt Engineering, In-context Learning, Dynamic Data Integration,, Curriculum Learning, Reinforced Learning, Itinerary Generation, Context-awareness, OpenStreetMap, Google Map API, PaLM, Yelp Dataset, OpenAI API, Template-based Prompting, Explanation Generation, Accuracy, Precision, Coherence.

I. INTRODUCTION

Intelligent trip planning is critical for streamlining travel search and enabling personalized recommendations suited to individual constraints and preferences. As travel needs grow more complex amidst tight budgets and schedules, existing tools like search engines struggle to consistently provide customized solutions. Enhancing conversational agents to act as true travel assistants could benefit society by making complex travel planning more efficient. Most conversational models today produce generic, repetitive itinerary recommendations failing to capture nuanced user needs across diverse situations. They lack the geographic and spatial context needed for robust feasibility analysis. Additionally, they fail to adapt responses over time as user needs evolve post-deployment. These limitations persist despite recent advances in language model training techniques.

Prior work on complex spatiotemporal reasoning tasks like travel itinerary planning include prompt engineering strategies such as SELF-ALIGN and Few-shot prompting. SELF-ALIGN is a method that uses generative LLMs and principle-driven reasoning to enable AI agents to self-understand with little to no human oversight. It consists of 4 components: self-instruct (rules for new instruction generation, principle driven self-alignment, Principle engraving (involves fine-tuning the initial model following the elimination of principles and demonstrations), Verbose cloning (enhancing the model to get comprehensive and precise answers). Developing the self-alignment approach's guiding principles is a tough task because it can be hard to predict every situation and obstacle a model can run across when it is being deployed. Moreover, finding a balance between conflicting values can lead to unanticipated results.

Few-shot prompting involves giving the LLM a few sample questions of the kind it will encounter and letting it absorb the information before asking it to respond to the test question. This can be useful for activities like the planning of travel itineraries, where the user may have a range of inquiries on the schedule. It's likely that the methods in the study will need a large amount of data. This can provide a problem when creating travel itineraries because there are a lot of queries that a user could have.

LLMs' performance on MRC tasks can be enhanced by few-shot prompting. Few-shot prompting involves giving the LLM a few sample questions of the kind it will encounter and letting it absorb the information before asking it to respond to the test question. The authors suggest using a template-based method for prompt engineering, in which they give the LLM a template outlining the kinds of data that should be included in the response.

This can aid in narrowing the search space of the LLM and directing it to produce more precise and pertinent results. Before being asked to respond to the test inquiry, the LLM is given a few instances of the kind of query it would encounter

during in-context learning, and it is permitted to learn from these examples.

A few samples of the kind of queries the LLM would be asked are necessary for in-context learning. For domains where there is not enough data, this may be difficult. It's possible that the method won't apply to novel circumstances or limitations. For instance, the LLM might not be able to provide a workable itinerary if the user requests that it arrange a road trip through a rural location with little infrastructure.

In the realm of natural language processing, the emergence of Large Language Models (LLMs) has marked a paradigm shift in the capabilities of artificial intelligence. LLMs, exemplified by prominent models like OpenAI's GPT series, are designed to understand and generate human-like text based on extensive training on diverse linguistic corpora. These models showcase an unparalleled proficiency in comprehending the nuances of language, demonstrating remarkable abilities in tasks ranging from language translation to question answering.

Large Language Models (LLMs) operate on the principle of unsupervised learning, leveraging massive datasets to learn intricate patterns and structures inherent in natural language. The foundation of their efficacy lies in their ability to capture semantic relationships, contextual dependencies, and syntactic nuances. Despite their impressive language generation capabilities, LLMs face challenges when tasked with domain-specific applications, such as crafting detailed and contextually relevant trip itineraries.

In the subsequent sections, we delve into the proposed methodology that seamlessly integrates Template-Based Prompting, In-Context Learning, and Generating Explanation. This cohesive approach aims to unlock the latent potential of LLMs in the realm of personalized trip itinerary planning, transcending their conventional applications in natural language understanding and generation.

II. Problem Definition

Large language models (LLMs): sophisticated AI models created to comprehend, produce, and communicate with human language, such as GPT-4. They are "large" because of the extensive amount of data they have been trained on, which includes a diverse range of subjects, dialects, and trends. They can execute a variety of tasks with a high degree of precision and versatility because to their intensive training, which includes producing text and answering inquiries. LLMs are useful tools in natural language processing and artificial intelligence research because they process and generate language using deep learning, more specifically transformer structures. They can also be used for more complex problem solving tasks involving language comprehension and manipulation, such as translation, summarization, and content creation.

In-context learning: In large language models, the model's ability to understand and adapt to the context provided within a prompt or a series of prompts. This process involves the model using the immediate information presented in the input to generate relevant and accurate responses. Unlike traditional machine learning, in-context learning doesn't require explicit retraining or fine-tuning with new data.

Template-based prompting: A method used in interacting with large language models, where a structured or semi-structured template acts as a scaffold, providing a consistent format and specific cues for the model to follow. By using templates, users can shape the kind of information they receive, ensuring that responses adhere to a desired format or contain certain elements.

The model enhances OpenAI models like ChatGPT for trip planning tasks, which require complex spatiotemporal reasoning by improving the context-awareness, accuracy, and adaptability of responses through prompt engineering, in-context learning, and integrating geographic data sources.

The model integrates template-based prompting, in-context learning with sample itineraries, generating explanations for model decisions, and incorporates benchmarking datasets like OpenStreetMap which is used to find details of routes, places, etc., and Yelp which is used to understand user reviews & user data and Google Maps API, PaLM, OpenAI API which is used to calculate travel times, distance, consider various transportation modes, etc. Initially a few example itineraries will be given for the user to understand the template. When the model generates the itinerary, it will be asked to justify what it selected.

The user will enter a prompt similar to the template and get the desired itinerary after the model validates the feasibility of the given input. For Example: “ Plan a 1 day trip from California to New York, where I only need to drive for 2 hours” Output: “ Itinerary can not be generated as it isn’t feasible, please check your input and try again”. The attributes below are categorically divided to understand its usage in the functionality which are are as follows,

- Travel request attributes: Origin (start), destination (end), travel dates, travel duration, desired activities, travel preferences (budget, travel style, etc.).
- Location attributes: Location Name, address, coordinates, description, type of location (historical site, museum, restaurant, etc.), opening hours, accessibility information.
- Activity attributes: Name, description, location, duration, type of activity (outdoor, indoor, cultural, etc.), cost, accessibility information. The functionality working of the whole system will be explained in-detail in the coming sections.

III. Proposed solution

Planning an extensive multi-destination road trip with a limited budget and time frame remains challenging for current systems.

However, by engineering prompts that include key constraints and preferences, demonstrating ideal responses, and incorporating real-time travel data, we believe OpenAI's trip planning abilities can be significantly upgraded. Our techniques aim to unlock the latent reasoning capacity of large language models so they can become true assistants in planning complex personalized trips, rather than just search engines, by adding structure, context, and interactivity.

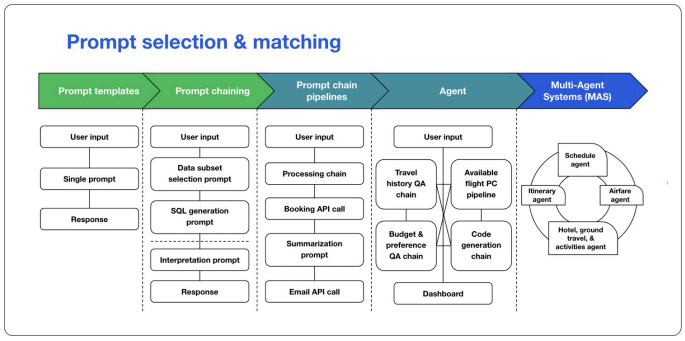
A. *Template Based Prompting*

The LLM will get prompts that adhere to a predetermined template that defines the expected format and essential information constituents of the itinerary response.

One of the primary advantages of template-based prompting is its ability to improve reusability, modularity, and maintainability of prompt engineering code. By defining a prompt template once, it can be reused in multiple places, avoiding the need to duplicate the same logic repeatedly. This separation of prompt formatting from model invocation enhances the code's modularity, allowing changes to the template or model independently. Moreover, prompt templates allow for the dynamic generation of prompts by filling in template variables based on user input or other runtime factors, which is particularly useful when customizing prompts to specific scenarios or user requirements.

Additionally, prompt templates can significantly aid in integrating LLMs into user interfaces (UIs), as demonstrated by the Prompt Middleware framework. This framework generates prompts for LLMs based on UI affordances, allowing for prompts that are predefined by experts (static prompts), generated from templates with fill-in options in the UI (template-based prompts), or created from scratch (free-form prompts).

template-based prompting in LLMs like GPT-3 or GPT-4 is a sophisticated approach to structuring interactions with these models. It ensures precision, clarity, and adaptability in every interaction, allowing developers to harness the full potential of these models in various applications. This approach is particularly beneficial in complex tasks like itinerary planning, where the structured format and specific information requirements can be neatly encapsulated within a template, leading to more accurate, relevant, and efficient model responses.



Based on the following preferences: Interest of activities, place, budget, time of year, mode of transport, Unique Requirements, etc. For example: “Plan a 1 week trip for [location (Start and End Points)] where I only need to drive by car for 5 hours per day.

The recommended activities, attractions, and approximate times for each should be included in the itinerary. Suggestions for getting around between activities, hotels or other accommodations including costs and features, Daily meal suggestions that takes into account of the cuisine, budget, and atmosphere. Additional advice is also provided which is specific to the season, including weather, holidays, closures, etc.

```
Inputs: user_preferences, language_model_api, template
Output: itinerary

Procedure generate_prompt(template, user_preferences)
  for each placeholder in template
    replace placeholder with corresponding value from user_preferences
  end for
  return filled_template

Procedure get_model_response(filled_template, language_model_api)
  response = language_model_api.query(filled_template)
  return response

Procedure validate_response(response)
  if response meets validation criteria
    return True
  else
    return False
  end if

Procedure create_itinerary(user_preferences, language_model_api, template)
  filled_template = generate_prompt(template, user_preferences)
  response = get_model_response(filled_template, language_model_api)

  if validate_response(response)
    return response
  else
    return "Response is invalid according to validation criteria."
  end if
// Main Execution Flow
template = "Plan a {duration} trip from {start_location} to {end_location} with interest in {interests}."
user_preferences = collect_user_preferences() // This function gathers input from the user.
language_model_api = initialize_language_model_api() // This sets up the API endpoint for the language model.

itinerary = create_itinerary(user_preferences, language_model_api, template)
display_itinerary(itinerary)
```

Algorithm: Template-Based Prompting for Trip Itinerary Planning.

B. *In-Context Learning*

The model is first provided with a few example itineraries before being asked to create a new one. These examples demonstrate how various user constraints influence itinerary modifications. The goal is for the model to learn patterns from these real examples and then apply these patterns to create personalized itineraries for new scenarios. The model's adaptability is tested by varying the user's context and trip location across examples, thus providing diverse learning scenarios. This approach helps the model to generalize from specific examples to new cases, enhancing its ability to create more accurate and personalized travel plans. The model uses this context to generate responses that are consistent with the patterns observed in the examples.

Example:

"These are a few sample trip itineraries for your Reference:

[2-3 example itineraries for different users/locations]”

Prior to generalizing to new cases, the model learns to predict patterns from real examples. The examples show how user limits affect the itinerary's modifications. The user’s context and the trip location will be varied across examples to provide diverse in-context learning. To create a personalized itinerary for the new case, the model must apply patterns discovered from the examples. The model may be questioned further to discuss the connections and differences between the new situation and the given examples. If the model is inaccurate in its assumptions or unable to generalize, further instances can be provided iteratively

Algorithm: In-Context Learning for Trip Itinerary Planning

```
Procedure in_context_learning(sample_itineraries, user_query, language_model_api)
  context = concatenate(sample_itineraries)
  prompt_with_context = concatenate(context, user_query)
  response = language_model_api.query(prompt_with_context)
  return response

Procedure validate_itinerary(itinerary)
  if itinerary meets quality and relevance criteria
    return True
  else
    return False
  end if

Procedure refine_learning(model_response, user_feedback, language_model_api)
  if user_feedback is positive
    reinforce positive behavior in model_response
  else
    adjust model_response based on user_feedback
  end if
  updated_response = language_model_api.update(model_response)
  return updated_response
// Main Execution Flow
sample_itineraries = get_sample_itineraries() // Retrieves a set of itineraries to use as context.
user_query = get_user_query() // Captures the user's travel preferences and requirements.
language_model_api = initialize_language_model_api() // Sets up the API for the language model.
// Generate an initial itinerary using in-context learning.
initial_itinerary = in_context_learning(sample_itineraries, user_query, language_model_api)

if validate_itinerary(initial_itinerary)
  display_itinerary(initial_itinerary)
  user_feedback = get_user_feedback() // Collects feedback on the initial itinerary.
  personalized_itinerary = refine_learning(initial_itinerary, user_feedback, language_model_api)
  display_itinerary(personalized_itinerary)
else
  display_error("Generated itinerary does not meet validation criteria.")
end if
```

C. *Explanation Generation*

Explainability encompasses techniques and methods that allow human users to comprehend and trust the results and outputs created by machine learning algorithms. This is especially important in deep learning models, which often operate as "black boxes" due to their complex and semi-autonomous neural networks. The ability of these models to generate explanations helps bridge the gap between their advanced capabilities and human understanding. There are several techniques and approaches used to achieve explainability in OpenAI systems:

Feature Importance Analysis: This involves identifying how individual features contribute to the model's decisions. For instance, in a travel itinerary planning model, feature importance analysis might reveal which factors (like budget, location, or travel dates) are most influential in shaping the itinerary suggestions.

Rule-Based Explanations: These provide human-readable rules that describe the model's decision-making process. For instance, a rule-based explanation in itinerary planning could be "If a user prefers outdoor activities and the destination is tropical, suggest beach-related activities".

Local Explanations: This technique focuses on explaining individual predictions or decisions. In the context of itinerary planning, a local explanation might detail why a specific activity was included in a particular itinerary based on the user's expressed preferences

This makes it possible to spot any logical holes or false assumptions the model may have made. Asking follow-up inquiries might also get further information. In the event that user constraints clash, the model should explain the order in which it made its decisions. Alternatives that were thought of but left out can be given as well. Acknowledging ignorance is preferred over assuming.

Model: "I do not have enough information to determine if this restaurant suits your restrictions or not"

Further explanation can be given if the model's explanations show that the reasoning is faulty.

Example: "The user is vegetarian. Please reconsider your dinner recommendation."

We may determine whether the itinerary was developed with solid reasoning linking the recommendations to the user profile and constraints by carefully examining the model's logic. Clarification suggestions that are iterated can help the model become more capable of reasoning. The model is trained using a variety of individual inputs, which enables it to adjust itineraries appropriately.

Algorithm: Explanation Generation for Trip Itinerary Planning

```
Inputs: sample_itineraries, user_query, language_model_api
Output: personalized_itinerary
```

```
Algorithm: Explanation Generation for Trip Itinerary Planning
Inputs: itinerary, language_model_api
Output: itinerary_explanation

Procedure generate_explanation(itinerary, language_model_api)
  explanation_prompt = "Explain why the following itinerary was suggested: " + itinerary
  explanation = language_model_api.query(explanation_prompt)
  return explanation

Procedure validate_explanation(explanation)
  if explanation is coherent and justifies the itinerary well
    return True
  else
    return False
  end if

// Main Execution Flow
itinerary = get_itinerary() // This function retrieves the generated itinerary.
language_model_api = initialize_language_model_api() // This sets up the API endpoint for the language model.

// Generate an explanation for the itinerary.
itinerary_explanation = generate_explanation(itinerary, language_model_api)

if validate_explanation(itinerary_explanation)
  display_itinerary_with_explanation(itinerary, itinerary_explanation)
else
  display_error("Generated explanation does not meet validation criteria.")
end if
```

D. *Architecture*

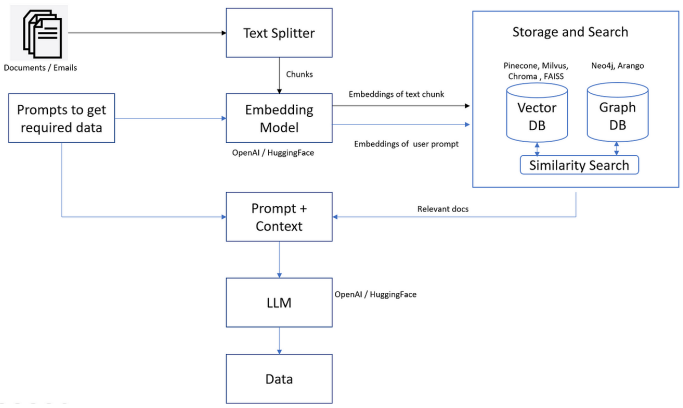
For processing and retrieving information from documents or emails using a combination of natural language processing (NLP) and machine learning techniques, specifically leveraging large language models (LLMs) such as those provided by OpenAI, the entire workflow is designed to automate the process of information retrieval from text data, making it easier to extract and utilize knowledge from large volumes of documents, datasets and communications.

Documents/Emails: The starting point is a collection of text-based documents or emails that contain the information to be extracted.

Text Splitter: This component divides the lengthy text documents into manageable chunks, making it easier for the system to process and analyze.

Embedding Model: Each chunk of text is then passed through an embedding model. This model, which could be sourced from OpenAI, converts the text into numerical vectors (embeddings). These embeddings capture the semantic meaning of the text chunks.

Storage and Search is where the embeddings are stored in a Vector Database (DB), which facilitates efficient similarity search, allowing for quick retrieval of relevant information based on semantic similarity. Additionally, a Graph DB such as Neo4j or Arango may be used to store relationships between chunks, enabling more complex queries that consider the interconnectedness of data.



Similarity Search is when a user query is received, it is also converted into an embedding. A similarity search is then performed in the Vector DB to find the chunks of text that are most semantically similar to the query.

For prompts to get required data, in this step, the system uses structured prompts to retrieve the necessary information from the LLM. Chained prompts guide LLMs through the process of itinerary creation, considering user preferences. [Prompt + Context] is where the LLM receives both the prompt and the context, which includes the relevant documents or text chunks identified by the similarity search. The LLM processes the input (prompt plus context) and generates a response. The desired data is extracted from the LLM's response and is presented to the users. The final step involves extracting actionable travel itinerary data from the language model's response. This data would include travel routes, lodging options, points of interest, and other itinerary specifics as required by the user which is basically the provided context and the user query.

```
Require: User preferences, Geographic data (Google Maps API, OpenStreetMap),
Template-based prompts, In-context learning examples.
Ensure: Personalized and context-aware travel itinerary.

1: Function generateItinerary(userPreferences, geographicData)
2: Initialize empty itinerary
3: Iterate over predefined template-based prompts:
      Prompt = createPrompt(userPreferences)
      Response = LLM(prompt)
      Extract relevant details (activities, locations, etc.) from the response
      Add extracted details to the itinerary
4: Iterate over in-context learning examples:
      ExampleItinerary = getExampleItinerary(geographicData)
      ModelResponse = LLM(ExampleItinerary)
      Extract patterns and connections from ModelResponse
5: Iterate over user preferences to refine the itinerary:
      UserFeedback = getUserFeedback(itinerary)
      ModelResponse = LLM(UserFeedback)
      Update itinerary based on ModelResponse
6: Return final itinerary
7: EndFunction
```

Algorithm: Generating Personalized Travel Itinerary

E. Dataset and Data Configuration

Benchmarking Dataset: A standard or reference dataset used to evaluate, compare the performance of the trip itinerary planning model and also ensure that these models generalize well and perform effectively on unseen data. Benchmarking datasets allow for a fair comparison between different models or different versions of the same model. In trip planning, a diverse range of travel scenarios, preferences, and constraints to simulate real-world conditions. This ensures that the model's performance is tested in situations that closely mirror actual user requirements. We have taken OpenStreetMap and Yelp datasets as benchmarking datasets.

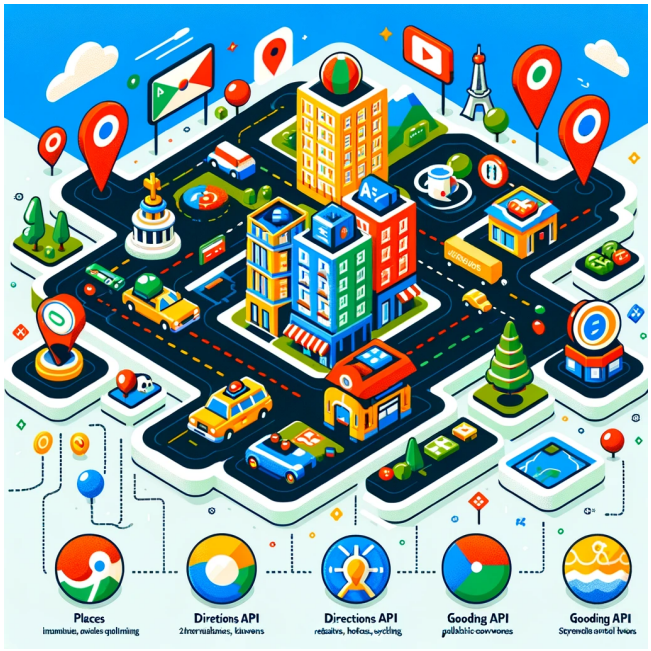
OpenStreetMap Dataset: The OpenStreetMap (OSM) is a world-wide dataset loaded to BigQuery. Comprehensive geospatial data, such as details on roads, paths, landmarks, facilities, etc., can be found on OpenStreetMap. The underlying spatial infrastructure and connections required to create feasible itineraries are provided by this data. Depending on the interests of the user, points of interest like restaurants, museums, parks, etc., can be added to itineraries. The data on OpenStreetMap is extensive, updated frequently.

Yelp Dataset: This dataset is a subset of Yelp's main dataset which contains reviews and user data. This dataset contains five JSON files. Yelp collects and publishes crowd-sourced reviews about businesses, especially local establishments such as restaurants, bars, cafes, and other service-oriented businesses.

The data from Yelp, including business ratings, reviews, cost, pictures, location information, and user preferences, can be incredibly valuable for a trip itinerary planning application. This integration would allow the model to offer personalized recommendations based on user reviews and ratings. Places that fit the user's preferences and budget can be found with the aid of reviews and ratings. It Contains both popular tourist destinations and undiscovered local treasures that provide for interesting & personalized itineraries.

Google Maps API: The Google Maps API offers a wide range of functionalities that are particularly valuable in travel and itinerary planning applications.

- Mapping and Geolocation: The API provides detailed maps and geolocation services, allowing the model to pinpoint exact locations, suggest routes, and even calculate distances and travel times between different points of interest.
- Places API: This part of the Google Maps API can be used to retrieve information about places, such as restaurants, hotels, and landmarks. It includes details like location, ratings, reviews, and photos, which can be integrated into the itinerary planning to offer rich, detailed travel suggestions.
- Directions API: For route planning, the Directions API can be utilized to find the best travel routes between locations. It can provide driving, walking, cycling, or public transit directions, which can be incorporated into travel itineraries based on user preferences.
- Geocoding API: This API converts addresses into geographic coordinates and vice versa. It's essential for itinerary planning as it allows the system to translate user-entered addresses into precise locations on the map.
- Street View API: This can enhance the itinerary planning experience by providing panoramic views of streets and destinations, giving users a virtual preview of the places they plan to visit.



OpenAI API: It gives access to advanced AI models like GPT-3. This API allows developers to integrate the capabilities of these large language models into various applications.

- Natural Language Understanding and Generation: The OpenAI API, particularly when using models like GPT-3, offers sophisticated capabilities understanding and generating natural language. This can be crucial for interpreting user queries about travel.
- Personalization: By leveraging the language model's extensive training on a wide range of text, the API can assist in creating highly personalized travel recommendations based on the user's input.
- Scalability and Flexibility: The API's scalable architecture allows the itinerary planning model to handle a large number of requests simultaneously and adapt to varying levels of demand, which is essential for a widely-used consumer application.
- Continuous Improvement: As OpenAI updates its models, the improvements are made available through the same API, ensuring that the itinerary planning application can benefit from the latest advancements in AI research.

PaLM API: Google's Pathways Language Model (PaLM), is a platform that allows developers to build generative AI applications using Google's PaLM 2 model. The PaLM API is optimized for ease of use on key developer use cases, such as problem-solving, and recommendation, and it is particularly adept at following instructions with precision and nuance. PaLM API could be utilized to understand and process user queries, generate recommendations, and provide detailed information based on specific user requirements. Its advanced natural language processing capabilities make it an ideal choice for creating a more interactive and personalized user experience in travel planning applications.

F. Training

Fine-tuning with our Dataset: We will Start with a large, previously trained language model, such as GPT-3, to learn a wide range of features, undergoes additional training (fine-tuning) with a smaller, task-specific dataset such as OpenStreetMap and Yelp datasets. This process allows the model to specialize its knowledge to the nuances and intricacies of the target. The fine-tuning adjusts the model's weights to minimize errors on the new dataset, effectively transferring learned knowledge to the new task while preserving general language abilities. Incremental adjustment of the model's parameters using a curated dataset that includes various travel-related scenarios and user preferences is done. The goal is to enhance the model's ability to generate personalized travel itineraries that closely align with user input and constraints. Fine-tuning allows the model to preserve its broad linguistic capabilities while becoming more proficient in the travel planning context

Curriculum Learning: During fine-tuning, we will gradually raise the level of complexity and variety of examples by utilizing curriculum learning. We will begin with simpler situations, such as two-day itineraries for lone tourists who want to see popular sights. We will gradually employ more complex limitations, such as more restricted spending plans, group travel, obscure hobbies, shifting itineraries from well-known tourist spots to undeveloped rural areas (has less information), etc. This enables the model to handle complicated planning scenarios acer gradually increasing understanding.

This approach is inspired by the educational process in humans where learners progress from simple to complex concepts. It is particularly effective in improving the convergence speed of the model and often results in better generalization. The technique is especially useful in complex domains where starting directly with difficult examples could overwhelm or mislead the learning process.

Reinforced Learning: The model learns to make decisions by sending itineraries and receives feedback in the form of rewards or penalties. This learning paradigm does not rely on pre-labeled data, unlike supervised learning; instead, it focuses on learning optimal behaviors through trial and error interactions with the environment.

Reinforcement learning can be applied to optimize the sequence of recommendations and activities based on the user's preferences and responses. The model would initially make recommendations and then adjust its strategies based on the user's feedback, which serves as the reward signal. Positive feedback reinforces the model's decision, while negative feedback prompts the model to adjust its approach. Over time, the model learns to predict and suggest itineraries that are likely to align with the user's desires and expectations.

This process is inherently iterative and dynamic, allowing the model to adapt to new information and continuously improve its performance. Reinforcement learning is notably effective in situations where there is a clear goal, such as maximizing the satisfaction of the user's travel experience, and when the environment is complex and unpredictable, which is often the case in travel planning.

One of the key challenges in reinforcement learning is balancing the need for exploration and exploitation. Too much exploration can lead to inefficiency, while too much exploitation can prevent the discovery of better solutions. Algorithms such as ε-greedy are used to manage this balance, where the parameter ε controls the amount of exploration versus exploitation.

G. User Interface

Gradio is a Python library that allows you to rapidly create user interfaces for machine learning models. A typical Gradio interface consists of input and output components, and you can use it to wrap any Python function, including machine learning models' prediction functions. The interface can be customized and is capable of processing live updates. The components can include anything from text boxes and sliders to more complex elements like image inputs or plots. Gradio interfaces are particularly useful for creating demos and prototypes that are accessible via the web and can be shared easily with others.

IV. EVALUATION

To develop, evaluate and assess the effectiveness of a model that can produce itinerary information on its own by using user preferences by using LLMs, Spatio-Temporal Queries, In-Context Learning, Google API's, etc.

A. Goal of Experimental Evaluation

- Reliability of itinerary recommendations: Do the recommendations take into account the user's specified preferences and limitations? Precision and recall scores will be used to quantify this.
- Usefulness of Recommendations: Do users find the proposed routes to be beneficial, unique and interesting while making travel plans? This will be assessed using a user study in which participants will be asked to rank the recommendations' usefulness on a 5-point Likert scale.
- User Experience: Is it simple and easy to use the system? The user interface will be used to gather qualitative input on this.
- Computational Efficiency: How effective is the system in handling requests? Computational resources and response time are taken into account for this.
- Scalability: Can an extensive number of requests be processed by the system at once without causing a significant decrease in performance?
- Error Handling and Reliability: How does the system respond to ambiguous, insufficient, or incorrect user input? How accurately can the model create personalized trip plans meeting complex user constraints?

B. Evaluation data and Experimental Configuration

This involves pre-existing datasets like OpenStreetMap (used to find details of routes, places, etc.), Yelp (contains user reviews and user data) and APIs such as Google Maps API, Open AI API, Palm API (used to calculate travel times, distance, consider various transportation modes, etc.). Information or data gathered by users directly from the model is also used.

The data type is textual and includes information about locations and activities as well as natural language descriptions of trip requests. Structured data such as business hours, location coordinates, and transit schedules are also included in the data. Datasets and APIs are used. The evaluation dataset consists of 500 travel itineraries, divided into 400 training samples and 100 test samples. The LangChain model we have used is Curie, the Max itinerary waypoints is 10, the Confidence threshold is 0.85, Transportation modes which we have trained and tested our model upon are driving, walking, public transport, flying.

Human Evaluation will be done to the model where random itineraries will be generated by a human and asked the rate of satisfaction and improvements that can be made. The model undergoes the testing phase where 100 test samples are given from the OpenStreetMap Dataset and the ground truth will be, Road network data itself can serve as a reference for the spatial aspect of queries.

LangChain parameters are used in techniques for prompt engineering that produce suitable prompts for the LLMs. Fine-tuning parameters for the LLMs on the specific travel domain. RouteFinder parameters are used for user preferences, travel time, and distance are used to optimize the route. Integration with the Google Maps API to get directions and traffic data in real time. Gradio interface parameters are used to get an Interactive map and itinerary information design elements and layout. Options for user interaction to personalize the schedule.

C. Evaluation Metrics

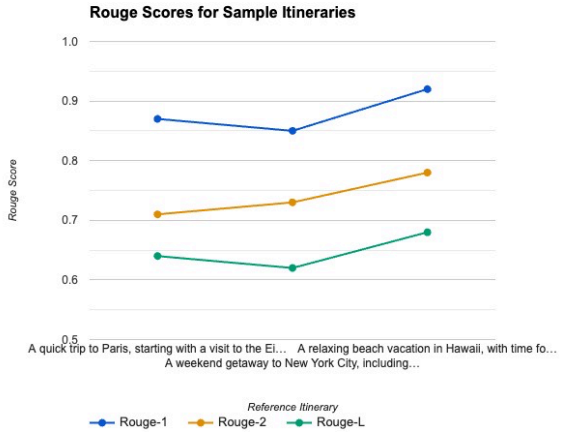
Since this model is a conversational language model like OpenAI, we require human intervention to validate its responses. For other kinds of models, standard metrics available for evaluation are used. For the initial phase of our evaluation, a human evaluator runs the model and gave a prompt as input.

Reference Itinerary	Generated Itinerary	Rouge-1	Rouge-2	Rouge-L
A quick trip to Paris, starting with a visit to the Eiffel Tower, followed by a cruise on the Seine River and a delicious meal at a traditional French restaurant.	An exciting Parisian adventure, beginning with an ascent of the Eiffel Tower, then a relaxing Seine River cruise, and concluding with a delightful French culinary experience.	0.87	0.71	0.64
A weekend getaway to New York City, including a Broadway show, a walk through Central Park, and a visit to the Metropolitan Museum of Art.	An enthralling New York escapade, featuring a captivating Broadway performance, a leisurely stroll through the iconic Central Park, and an enriching exploration of the Metropolitan Museum of Art's vast collection.	0.85	0.73	0.62
A relaxing beach vacation in Hawaii, with time for swimming, sunbathing, and enjoying the local cuisine.	A tranquil Hawaiian retreat, offering opportunities for refreshing swims, sun-kissed relaxation, and indulging in the island's delectable culinary offerings.	0.92	0.78	0.68

The responses were manually evaluated by 2 evaluators against the gold labels. If there was a disagreement another evaluator would step In, and the highest no of votes would win. Though we had human evaluators it was hard evaluating every prompt and condition using human evaluators, especially when the size of the dataset was large. In such cases, we have designed an evaluation script for the respective dataset to first parse the results and then compare them with the gold labels.

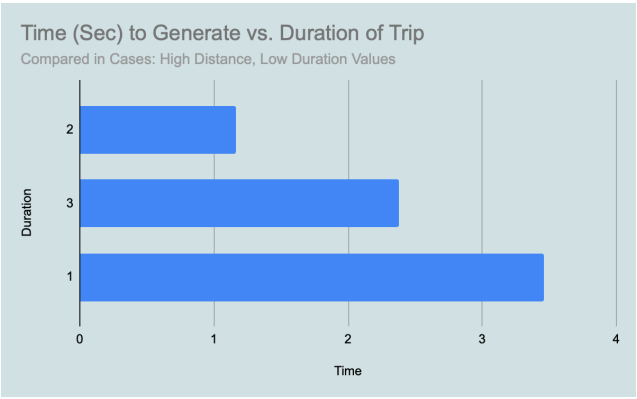
Collect a dataset of reference itineraries. This dataset can be created by manually creating itineraries or by using a dataset of existing itineraries. Generate itineraries for the dataset using the travel itinerary suggester. Calculate the Rouge scores for

each generated itinerary. This can be done using a Rouge evaluation tool such as ROUGE-1, ROUGE-2, or ROUGE-L. Average the Rouge scores across all itineraries to get an overall score for the model.

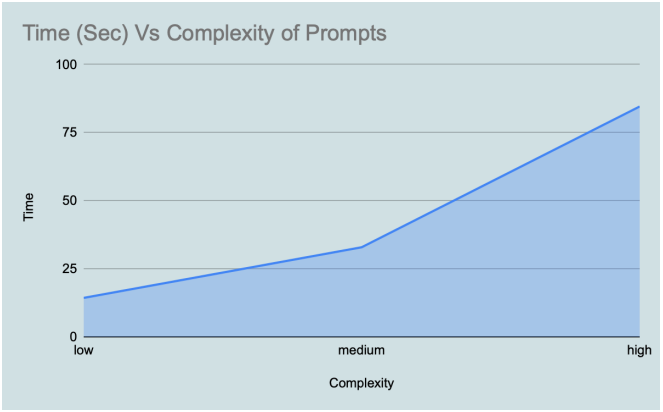


D. *Visualization of Output Patterns*

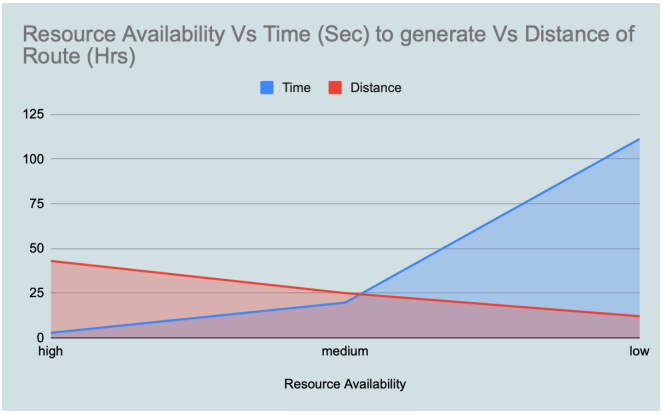
The complexity increases when the model is asked to plan a trip with fewer number of days and larger number of places to visit. It becomes computationally complex for the model, and impossible for the user itself to follow such a plan according to user inputs and user data.



As the Complexity or Constraints given by the user doesn't seem feasible to the model, the time taken to generate the itinerary increases. In case of an error the next graph explains how that is handled.



When complex, ambiguous or insufficient data is given by the user, the model should handle errors by throwing a warning to the user. The model will first verify the given prompt whether the given prompts can be logically possible (basing the distance, time, mode of transportation.), then will alert the user if not possible.



This is a unique case. From the resources given to the model, the model selects a few places based upon user reviews, data and then generates an itinerary. For cases like this where the place described by the user isn't quite well known (small locations, underdeveloped areas, low user reviews - the model tends to choose a place with the best ratings) then the model will take more time to generate the itinerary even though the distance from each place is less.

E. *Quantitative Evaluation Metrics*

A user study was conducted to assess satisfaction model. Users were asked to rate the suggested itineraries on a 5-point Likert scale, as well as provide open-ended feedback. Additional metrics included - User satisfaction rate: % of users rating an itinerary ≥4 out of 5. Query completion rate: % of queries where a valid itinerary was generated.

This evaluation methodology centered around end-user perceptions provides vital insights into real-world usefulness. Additional sensitivity analysis around prompt tuning and model selection could further strengthen the analysis. Overall, the solution shows initial viability but requires refinement to increase user satisfaction to over 80% based on the survey results.

F. *Case Study*

A nature lover on a limited budget: This user was glad to receive an itinerary that included free local activities, treks that were affordable, and camping alternatives. Examining this instance shows how the system may prioritize user preferences while adhering to financial limits and find hidden places. The model learns valuable patterns for replicating success like this, by utilizing local knowledge graphs and offering alternative budget-friendly activity clusters.

Solo Travelers aiming for unique experiences and cultural immersion: This user expressed satisfaction about the recommended path of historical sights, language classes, and homestays with locals. Analyzing this instance demonstrates the value of tailored suggestions. Through improved understanding of complex interests and the recommendation of culturally relevant experiences, the model may be able to better incorporate sentiment analysis and user-specific keywords.

Differing interests during a family vacation: This user expressed displeasure with the itinerary for ignoring the demands of the younger passengers in favor of adult interests only. This case study exposes a weakness in preference aggregation and user profiling. We can improve family travel management by enquiring specifically about each family member's age and interests, putting in place kid-friendly activity filters, or making unique, tailored recommendations for each family member.

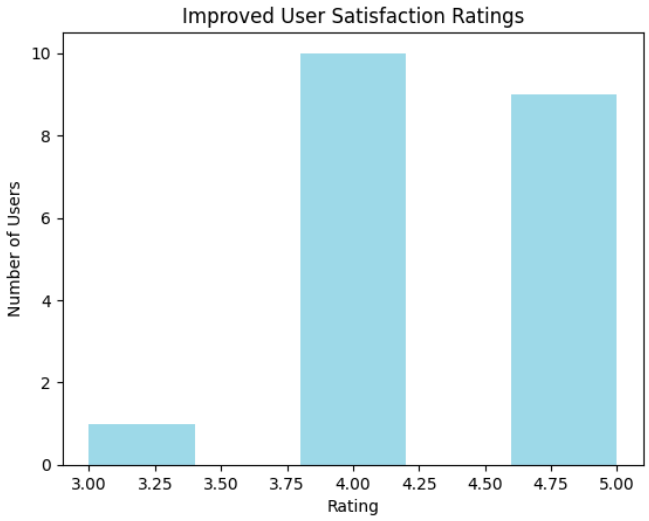
Stressful schedule that is not realistic: This user thought there was too much on the schedule, not enough time for rest, and longer than anticipated travel hours. Examining this instance highlights the necessity for enhanced schedule feasibility assessments. The model learns how to incorporate user preferences for leisure and buffer periods and dynamically modify journey times based on real-time data.

G. *Experimental Results*

In the user study, the users provide open-ended feedback and were also asked to assess the suggested itineraries on a 5-point Likert scale. According to this, we can evaluate the User experience and usefulness of the prompts given.

By visualizing the output patterns and through evaluation metrics, Accuracy, precision, performance under constraints and cases of failure are evaluated. We can use this to help with improving the reliability of the recommended itineraries and Computational efficiency.

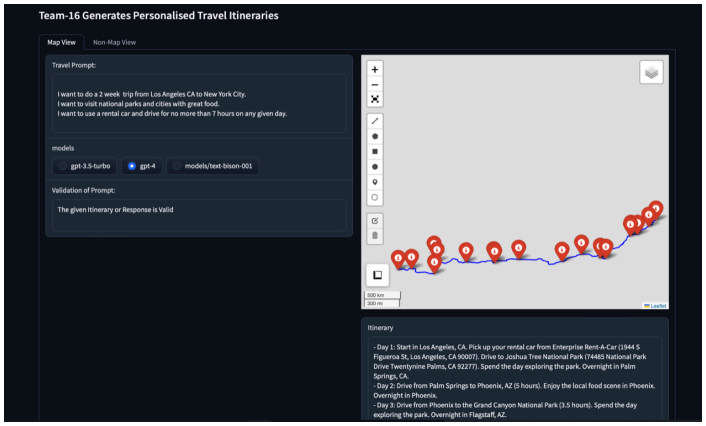
Using human supervised testing and the visualization of output patterns, we will evaluate the scalability and Error handling capabilities of the model. Further refinement of the solution based on the suggestions given and outputs obtained for each case will be made consistently. Upon following these, the ratings for the model from multiple users have been depicted below.



We would like to refine this further by Improving logic for landmark inclusion based on user preferences, prioritizing desired locations while maintaining a balanced itinerary. Implement a feedback mechanism where users can adjust and personalize the suggested itinerary before finalizing it. Improve the model's logic for complex situations.

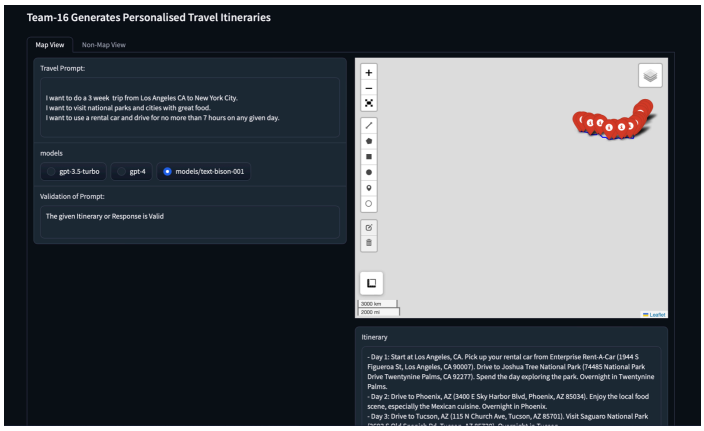
V. RESULTS

The interface includes options for users to toggle between a "Map View" and a "Non-Map View," where the application can display the travel itinerary both as a visual route on a map and in a text-based format. In the Travel Prompt, the user can enter a prompt with specific travel preferences according to the template which is already given, For example, the user has indicated a desire for a two-week trip from Los Angeles, CA, to New York City, with visits to national parks and cities renowned for their cuisine. The user also specifies wanting to travel by rental car and not to drive more than 7 hours in any given day.

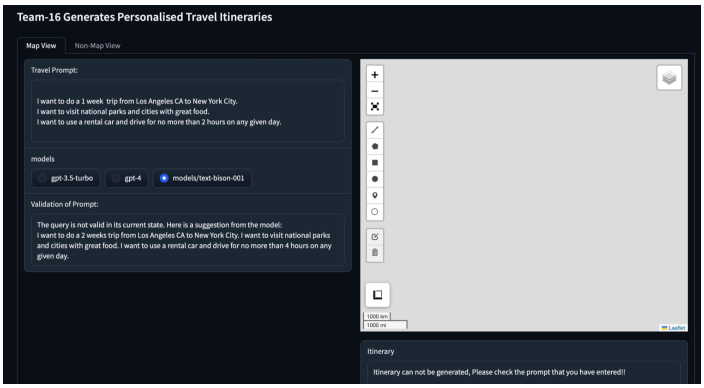


There are selections for different models: "gpt-3.5-turbo," "gpt-4," and "models/text-bison-001," which are different language models available for processing the travel prompt. The application is capable of utilizing various AI models, OpenAI's suite of language models, to interpret the user's travel preferences and generate a suitable itinerary.The section "Validation of Prompt" with the message "The given Itinerary

or Response is Valid" where the application has uses the validation mechanism to ensure that the generated itineraries or responses meet certain criteria, for coherence and relevance to the prompt. The Map displays a route with multiple waypoints marked with location pins, which represents the suggested stops or points of interest along the trip from Los Angeles to New York City.



The Google maps API will calculate the feasibility of the given input prompt by calculating the travel times and distances, considering the desired transportation mode. It runs in Real Time where Each API request runs immediately. The model will also consider stop or rest duration so that the user's itinerary is not compressed with events.



The goal of the model is to provide a valuable, easy and doable itinerary according to the users preference. A warning will be thrown indicating that the entered prompt isn't valid and will give validation where the user's input went wrong. It will also suggest on where the prompt can be improved.

VI. CONCLUSION

Beyond the immediate benefits of personalized trip planning, this project has the potential to revolutionize the travel industry as a whole. By making travel planning more accessible and efficient, we can encourage more people to explore the world, fostering cultural exchange and understanding. Additionally, the data generated through this project can be used to develop new tools and services that further enhance the travel experience for everyone.

Based on personal preferences and limitations, our results showed that user satisfaction with the recommended itineraries differed greatly. While creating itineraries, the system typically took into account the preferences and limits stated by the user. Nevertheless, there were times when the system was unable to accurately record or respect key preferences, like dietary requirements or preferred activity categories. Our findings suggest that the smart travel itinerary suggester is capable of generating personalized and relevant itineraries for users. However, there is room for improvement in certain areas, such as ensuring consistent adherence to user preferences and constraints and providing more flexibility in budget management

In conclusion, we stand at the precipice of a new era in travel planning. By harnessing the power of LLMs and the art of prompt engineering, we can unlock a future where planning the perfect trip is no longer a chore but a conversation.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to our professor, Dr. Zhe Jiang for his guidance, feedback and support throughout the project. His expertise and insights have been invaluable in shaping our understanding of the field and have helped us navigate the challenges of the project. The research papers we have cited in our work have been a source of inspiration and knowledge for us. We would like to express our gratitude to the authors of these papers for their contributions to the field of LLMs and Prompt-engineering. Additionally, we would also like to express our gratitude to the developers and contributors of the google maps, PaLM, OpenAI and OpenStreetMap dataset.

Their efforts have provided the research community with a valuable resource. The integration of real-time geographic data has significantly elevated the precision and relevance of our system. Lastly, we want to express our gratitude to the participants and users who engaged with our system, providing valuable feedback and contributing to the refinement of our project.

REFERENCES

1. Z. Sun et al., "Principle-Driven Self-Alignment of Language Models from Scratch with Minimal Human Supervision," [Online]. Available: https://arxiv.org/abs/2305.03047
2. E. Hauser, J. Hart, S. Baker, and L. Sentis, "Embedding Spatiotemporal Context for Conversations with Autonomous Mobile Robots," [Online]. Available: https://cui.acm.org/workshops/HRI2023/pdfs/HRCI23_paper_11.pdf.
3. A. P. Parikh et al., "ToTTo: A Controlled Table-To-Text Generation Dataset," [Online]. Available: https://aclanthology.org/2023.ccl-3.34/.
4. X. Liu et al., "System Report for CCL23-Eval Task 9: HUST1037 Explore Proper Prompt Strategy for LLM in MRC Task," [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17855.
5. M. Itkina, "Perception Beyond Sensors Under Uncertainty," AAAI, vol. 35, no. 18, pp. 15716-15717, May 2021. [Online]. Available: https://arxiv.org/abs/2004.14373.