

Announcements

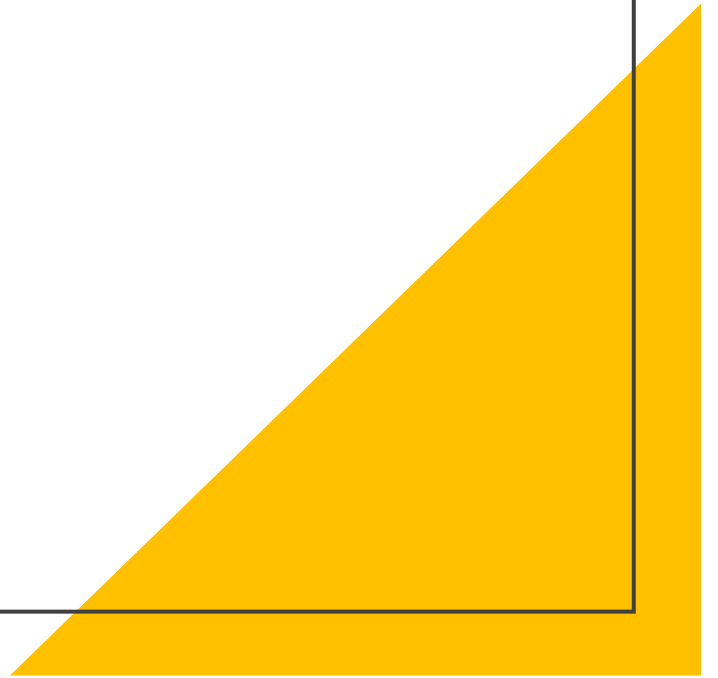
- Team Formation CATME Survey Due Date 8/25.
- Sustainability Survey and AI survey extra credit due 8/26. Please fill the survey out.
- Office Hours will begin on 8/26 – in CCB 267
- All Project 1 related assignments have released
 - Planning Assignment
 - Progress Report
 - Requirements
 - Presentation, Demo, Code, Report
 - Design
- Project Teams and Mentors will be assigned by 25th

CS 3300 A Introduction to Software Engineering

Lecture 02: Project Description; Prompt Engineering

Dr. Nimisha Roy ▶ nroy9@gatech.edu

Project Description



LocationSearch Web Application using SpringBoot,
Google Cloud Platform and AI tools

Problem Statement

Give me all interesting nearby locations (could be restaurants/tourist attractions/hotels/parks etc.) around a given coordinate. You need to have 2 filters for the type of location

LANDING PAGE

LocationSearch

This app returns all locations of interest near a specified coordinate

 [Not a user?](#)

LANDING PAGE

LocationSearch

This app returns all locations of interest near a specified coordinate

 [Already a user?](#)

Problem Statement

USER PAGE

Select a Search Area

Latitude:

Longitude:

Search radius:

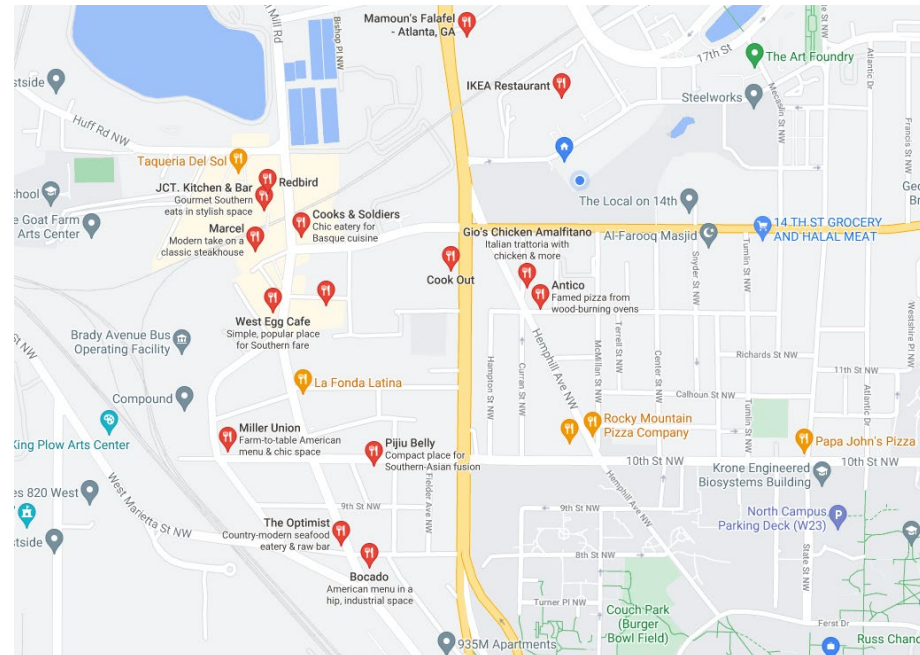
Find Places!

Search Results

Show entries

Search

Name	Distance to Point (ft)	Address	Rating	Price Level
JCT Kitchen & Bar	1815	1198 Howell Mill Rd #18, Atlanta, GA 30318	4/5	\$\$
Piju Belly
Miller Union
West Egg Cafe



Log Out

Required Features

- MMF1 - Landing Page / User Authentication
 - Landing Page: A welcoming interface that briefly describes the functionality of the application.
 - Sign Up/Login: User authentication system allowing users to sign up for a new account and log in. Utilize OAuth for secure authentication.
- MMF2 – Basic Input & Output - Map Display:
 - Initial Display: Upon login, users see a dynamic map where they can input latitude, longitude, and radius to search for places.
 - Input Fields: Allow users to enter latitude, longitude, and radius through user-friendly input forms with validation to ensure data integrity.
- MMF2 – Basic Input & Output - Search Results and Details:
 - Results Display: After entering search criteria, the map updates to show pins representing locations that fit the user's search parameters.
 - Detailed List View: A sidebar or overlay listing detailed information of places such as distance from the input coordinate, user ratings, operating hours, and price level.
- MMF3 – Filtering and Sorting - Location Filters:
 - Type Filters: Two selectable filters allowing users to specify the type of locations they are interested in, such as restaurants, tourist attractions, hospitals, etc.
- MMF3 – Filtering and Sorting - Advanced Filtering and Sorting Options:
 - Sorting Functionality: Users can sort the list of locations by various criteria like distance, ratings, or price level.
 - Number of Places Displayed: Allow users to choose how many locations to display on the screen at one time.
- MMF4 - Responsive and Interactive Map:
 - Hover Details: When a user hovers over a location pin on the map, show a mini-popup with key details like name, distance, and user rating.
 - Selection and Details: Clicking on a location pin or list item brings up a detailed modal or pane showing more extensive information, with options to get directions or visit the website.

Technology Suggestions

GOOGLE CLOUD PLATFORM*

Offers services for compute, storage, networking, big data, machine learning and the internet of things (IoT), as well as cloud management, security and developer tools.

BACKEND SERVER ARCHITECTURE

- **SpringBoot*** – Framework for developing the backend server architecture and to run the server application easily
- **Maven*** - Used for dependency injection and management as a Java package manager. Provided as a part of central repository in Spring
- Mockito – Test stub method for unit testing SpringBoot application
- VS Code/IntelliJ - IDE

DATABASE

- GCP DataStore
- MySQL
- Google Storage

* **Must use**

Technology Suggestions

API

- Google Places API - To implement an interactive, searchable map
- Google Distance Matrix API/ OpenStreetMap API

FRONTEND

- Thymeleaf - Java template engine for both web and standalone environments. Set of Spring integrations.
- Leaflet - Open-source JavaScript library for creating interactive maps
- jQuery - JS Library for implementing HTML document traversal and manipulation, event handling, animation, and Ajax.
- Bootstrap – CSS framework to create modern websites and web apps
- React.JS - open-source JavaScript library used for building user interfaces for single-page applications

VERSION CONTROL

GitHub*

AI Tools for Implementation

GPT 4o API*

Copilot

* Must use

Some Considerations/Functionalities

Password Security

- Salting & Hashing Passwords
- JWT – Validity of Tokens (optional)

Database

- Google DataStore is recommended.
- Google storage has some problem: 2 people simultaneous logging in may cause overwrite

Webjars

Use webjars - client-side web libraries packaged into JAR (Java Archive) files to easily manage web dependencies

Testing

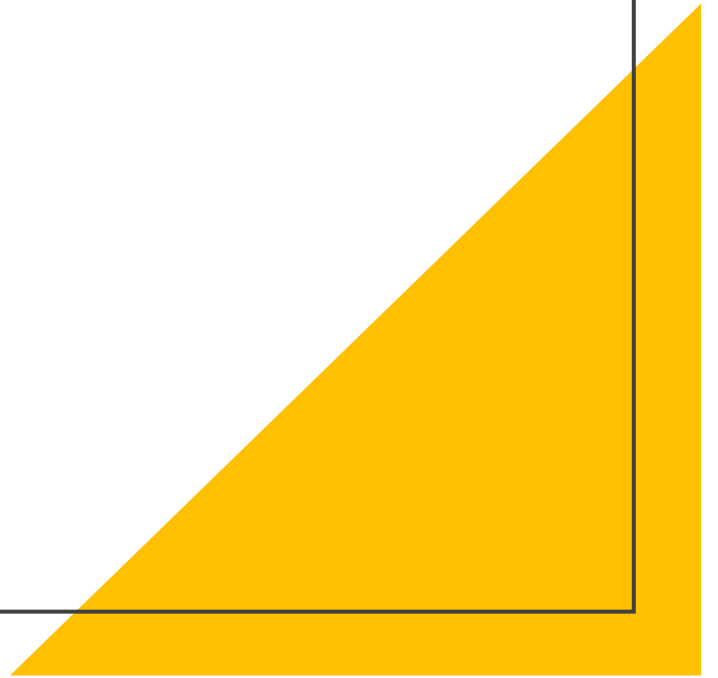
Important. Modules Included in Google App Engine.

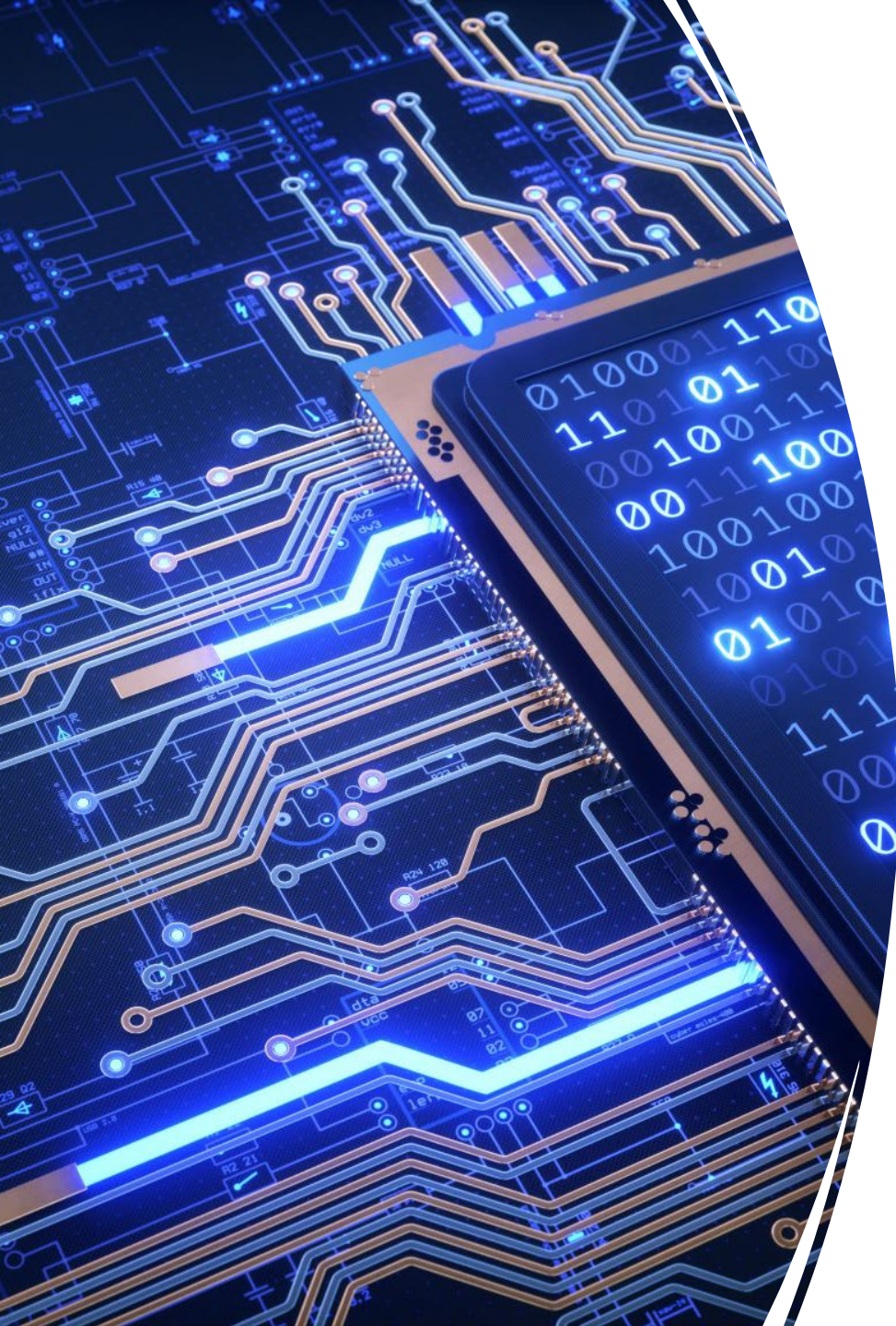
Project 1

Progress Report



Prompt Engineering





Introduction to Prompt Engineering in Software Engineering

- **Prompts** involve instructions and context passed to a language model to achieve a desired task.
- **Prompt engineering** is the practice of developing and optimizing prompts to efficiently use large language models (LLMs) for a variety of applications.
- **Importance in Software Engineering:** Accurate prompts can significantly enhance the utility of AI in code generation, debugging, requirement elicitation, and documentation, making them indispensable in modern software development.

Fundamentals of Prompt Engineering

Zero-shot Prompting: Instructing the AI without any context-specific training; based purely on general model training.

Few-shot Prompting: Providing examples to AI to specify the context or format of the desired answer; helps the AI understand domain-specific tasks.

Chain-of-Thought Prompting: Constructing prompts that simulate a logical reasoning path for the AI, helping it tackle complex software engineering problems step-by-step.

Prompt Chaining: Using the output of one prompt as the input for another, effectively chaining prompts together to achieve more complex tasks

Zero Shot Prompting

- Zero-shot prompting involves instructing the model to perform a task without providing examples, background, or additional information.
- Let's demonstrate zero-shot prompting via an example:

Query: Generate a python code snippet to get the average grade of students in homework 1

Output: Sure, here's a Python code snippet that calculates the average grade of students for homework 1. This assumes you have a list or array of grades for each student.

```
# Sample list of grades for homework 1
grades_homework1 = [85, 92, 78, 90, 88, 84, 70, 95, 82, 87]
# Calculate the average grade
average_grade = sum(grades_homework1) / len(grades_homework1)
print(f"The average grade for homework 1 is: {average_grade}")
```

Few Shot Prompting

- Few-shot prompting involves providing examples in prompts to guide the model towards better performance.
- **Example from Brown et al. 2020:**

Query: Correctly use a new word in a sentence.

Example 1: A "whatpu" is a small, furry animal native to Tanzania. An example of a sentence that uses the word whatpu is: "We were traveling in Africa, and we saw these very cute whatpus."

Example 2: To do a "farduddle" means to jump up and down really fast. An example of a sentence that uses the word farduddle is:

Output: When we won the game, we all started to farduddle in celebration.

Few Shot Prompting: Another Example

- **Query:** Generate a function that can calculate and return a response status for an API call.
- **Example 1:** A function `getResponseStatus` returns "Success" if an API call response status is 200, and "Error" otherwise. An example of this function is

```
def getResponseStatus(response_code):  
    return "Success" if response_code == 200 else "Error"
```

- **Example 2:** The "checkApiCall" function is designed to make an API call and use `getResponseStatus` to determine and print the status.
- **Output:** Here's a Python code snippet that calculates and returns the response status for an API call:

Few Shot Prompting: Another Example

- **Output:** Here's a Python code snippet that calculates and returns the response status for an API call:

```
import requests

def getResponseStatus(response):
    if response.status_code == 200:
        return "Success"
    elif response.status_code == 404:
        return "Not Found"
    else:
        return "Error"

def checkApiCall(url):
    try:
        response = requests.get(url)
        status = getResponseStatus(response)
        return f"API Call Status: {status}"
    except requests.exceptions.RequestException as e:
        return f"Failed to make API call: {str(e)}"

# Example usage
url = "https://api.example.com/data"
result = checkApiCall(url)
print(result)
```

Chain of Thought (CoT) Prompting

Prompting can be further improved by instructing the model to reason about the task when responding

- This is very useful for tasks that requiring reasoning
- You can combine it with few-shot prompting to get better results
- You can also do zero-shot CoT where exemplars are not available

Chain of Thought (CoT) Prompt Example

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27.



Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5+6=11$. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had $23-20=3$. They bought 6 more apples, so they have $3+6=9$. The answer is 9.



Prompt Chaining

- Prompt Chaining is a technique where the output of one prompt serves as the input or context for the next, creating a sequence of interlinked prompts that guide the AI towards a complex goal
- Enables the handling of multi-step problems within AI systems by maintaining a contextual flow, improving accuracy in tasks requiring deep contextual understanding
- This method allows for more nuanced interactions with AI, closely mimicking human-like reasoning and procedural task management
- **Example:**
 - **Initial Prompt:** "What is the first step in making a cake?"
 - **AI Output/Next Prompt:** "Gather all necessary ingredients."
 - **Continuation:** "What is the next step after gathering ingredients?"
 - **AI Output/Next Prompt:** "Preheat the oven to the appropriate temperature."
 - **Final Output:** "Mix the ingredients and place them in the oven."

Prompt Chaining Example

Initial Prompt:

- **Query:** "Generate a Python function to parse and summarize JSON data from a project management tool's API."
- **Chain Prompt 1:**

Output from Initial Prompt:

```
def parse_json(json_data):  
    """Extracts project details from JSON data."""  
    projects = json.loads(json_data)  
    summary = {project["id"]: project["name"] for project in projects}  
    return summary
```

Prompt Chaining Example

- **Follow-up Query:** "How would you use this function to display a formatted list of project names and their statuses?"
- **Chain Prompt 2:**
- **Output from Chain Prompt 1:**

```
def display_projects(json_data):
    projects = parse_json(json_data)
    for project_id, project_name in projects.items():
        print(f"Project ID: {project_id}, Project Name: {project_name}")

# Example JSON Data
json_data = '''
[
    {"id": 1, "name": "Project Alpha", "status": "Active"},
    {"id": 2, "name": "Project Beta", "status": "Completed"}
]
...

display_projects(json_data)
```

Prompt Chaining Example

- **Follow-up Query:** "Now, create a function that updates the status of a specific project by project ID in the project list."
- **Chain Prompt 3:**
- **Output from Chain Prompt 2:**

```
def update_project_status(projects_json, project_id, new_status):
    projects = json.loads(projects_json)
    for project in projects:
        if project['id'] == project_id:
            project['status'] = new_status
            break
    return json.dumps(projects)

# Updated usage
updated_json = update_project_status(json_data, 1, 'Completed')
print(updated_json)
```


Techniques for Effective Prompt Engineering

- **Clarity and Specificity:** Ensuring prompts are clear and detailed reduces ambiguity and improves AI accuracy. Example: Specify software frameworks, expected outputs, error handling needs.
- **Incorporation of Keywords:** Embedding technical terms specific to software development can help focus AI's responses on the relevant area. Example: "API", "async function", "state management".
- **Iterative Refinement:** Adjusting and evolving prompts based on the quality of AI's responses to hone in on the optimal solution.

Practical Applications of Prompt Engineering in SE

- **Code Generation/Completion:** Crafting prompts that describe the exact functionality and parameters of the code to be generated, potentially including compliance with coding standards and performance metrics.
- **Automated Testing:** Automatically generating unit tests, integration tests, and system tests that ensure all aspects of an application are thoroughly validated.
- **Debugging Assistance:** Formulating prompts that help identify bugs by describing the malfunction and the expected behavior, thus guiding AI to propose plausible fixes.
- **Requirements Specification:** Automating the generation of requirement specifications documents from meeting notes or user interviews.
- **UML Diagram Generation for Software Design:** creating class diagrams, sequence diagrams, or state diagrams from project documentation, meeting transcripts, and architect explanations

Navigating Challenges in Prompt Engineering

- Common Pitfalls: Overly broad prompts leading to generic outputs, under-specification causing irrelevant solutions.
- Best Practices:
 - Always start with a clear, concise objective; incorporate feedback loops to refine prompts with more detailed prompts; use structured prompt templates where possible.
 - Don't believe in all outputs. Use your coding knowledge to analyze and adapt the output.