**INTEL® UNNATI INDUSTRIAL TRAINING PROGRAM 2024**

**CUSTOMIZED AI KITCHEN FOR INDIA**

PROJECT REPORT



**MENTOR:**

**MRS.SHIFANA RAYESHA**

ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

B.S.ABDUR RAHMAN CRESCENT INSTITUTE OF SCIENCE AND TECHNOLOGY

* **TEAM MEMBERS AND CONTRIBUTIONS**
  + - **ABDUL HAADI N**
    - **DHARUN SUVADHAGAN S**
    - **ASARAF BASHA**
    - **ASWIN KUMAR M**
    - **ERINA SMILIN**

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **S NO** | **TOPIC** | **PG NUMBER** |
| **1** | **ACKNOWLEDGEMENT** | **1** |
| **2** | **INTRODUCTION** | **2** |
| **3** | **SYSTEM DESIGN AND ARCHITECTURE** | **6** |
| **4** | **IMPLEMENTATION DETAILS** | **8** |
| **5** | **SIMULATION PROCESS** | **16** |
| **6** | **RESULTS AND ANALYSIS** | **21** |
| **7** | **DISCUSSION** | **23** |
| **8** | **CONCLUSION** | **27** |
| **9** | **REFERENCES** | **28** |
| **10** | **APPENDICES** | **29** |
| **11** | **GLOSSARY OF TERMS** | **30** |

**ACKNOWLEDGMENT**

We extend our heartfelt gratitude to MRS.SHIFANA RAYESHA Assistant Professor in the Department of Computer Science and Engineering at B.S. Abdur Rahman Crescent Institute of Science and Technology, for her exceptional guidance, unwavering mentorship, and invaluable support throughout the entire journey of developing CUSTOMIZED AI KITCHEN. Her expertise, encouragement, and insightful feedback have profoundly influenced the project's success and our professional growth.

We also express sincere thanks to Dr. Sharmila Sankar, Dean of the Department and,

Dr. Aisha Banu W Head of the Department of Computer Science and Engineering for their continuous support, encouragement, and belief in our capabilities. Their leadership and vision have provided the necessary framework and resources that have been instrumental in realizing this project.

Furthermore, we are grateful to the team of the Intel Unnati program for their steadfast support, valuable resources, and enriching opportunities that have significantly contributed to the advancement and success of this endeavour. Their guidance and contributions have played a pivotal role in shaping our project and expanding our knowledge in the realm of technology-driven solutions.

Our heartfelt appreciation also goes to B.S. Abdur Rahman Crescent Institute of Science and Technology for fostering an environment conducive to innovation and learning. The institute's commitment to excellence and academic rigor has provided us with the foundation to explore and implement cutting-edge technologies in real-world applications.

This project has been a journey of learning, growth, and collaboration, and we are immensely grateful to all who have supported us along the wa

**Solution approach:**

Certainly! Here’s a simplified explanation of the solution approach for your "Customized AI Kitchen for India" project:

Solution Approach:

1. Collect User Input:

- \*Step\*: Prompt the user to select a cuisine type from a list (e.g., Indian, Italian, Chinese).

- \*Step\*: Ask the user to enter the ingredients they have available.

2. \*Load AI Model and Tokenizer\*:

- \*Step\*: Use the Hugging Face Transformers library to load a pretrained model and tokenizer designed for recipe generation.

3. \*Generate Recipe Suggestions\*:

- \*Step\*: Combine the user's selected cuisine and ingredients into a single input prompt.

- \*Step\*: Use the model to generate a recipe suggestion based on this prompt.

4. \*Display Recipe Suggestion\*:

- \*Step\*: Show the user the generated recipe suggestion.

5. \*Generate Detailed Cooking Instructions\*:

- \*Step\*: Use the model again to generate detailed cooking instructions for the suggested recipe.

- \*Step\*: Ensure the instructions are clear and easy to follow.

6. \*Present Final Recipe\*:

- \*Step\*: Display the detailed cooking instructions to the user.

Key Tools Used:

- Google Colab: To run the code and display outputs.

- Hugging Face Transformers: To load and use the AI model and tokenizer.

- Python: The programming language used for coding the entire solution.

Implementation Details:

- Model Loading: Load the model and tokenizer from Hugging Face.

- Input Handling: Collect user inputs for cuisine type and ingredients.

- Recipe Generation: Generate a recipe suggestion using the model.

- Instruction Generation: Generate detailed cooking instructions for the suggested recipe.

- Error Handling: Provide clear error messages for invalid inputs or other issues.

This simplified solution approach should help you explain your project clearly and concisely. If you need more specific details or further assistance, feel free to ask!

# IMPORTANCE OF THE PROJECT:

# The "Customized AI Kitchen for India" project holds significant importance due to several key reasons:

# 1. \*Cultural Relevance\*: Tailored specifically for Indian cuisine, it respects and celebrates the rich culinary traditions and diversity of India. This focus ensures that users receive recipe suggestions and cooking instructions that align with their cultural preferences and local ingredients.

# 2. \*Enhanced Culinary Exploration\*: By leveraging AI and machine learning, the project facilitates exploration of a wide range of Indian dishes. Users can discover new recipes, experiment with different flavors, and expand their culinary repertoire, ultimately promoting creativity and innovation in home cooking.

# 3. \*Accessibility and Convenience\*: The automation of recipe suggestion and generation simplifies meal planning and cooking processes. Users can easily find recipes based on available ingredients, dietary preferences, and cultural preferences, making home cooking more accessible and enjoyable.

# 4. Educational Value: Serving as an educational tool, the project helps users learn about traditional Indian ingredients, cooking techniques, and regional variations. It promotes culinary learning and appreciation, especially among those new to Indian cuisine.

# 5. Community Engagement: Facilitates sharing and collaboration among cooking enthusiasts and communities interested in Indian food. Users can share their favorite recipes, cooking tips, and experiences, fostering a sense of community around food and culinary traditions.

# 6. Technological Innovation: Demonstrates the application of advanced AI and NLP technologies in everyday tasks like recipe recommendation systems. It showcases how technology can enhance daily life by simplifying tasks and providing personalized experiences.

# 7. Potential for Expansion: Beyond recipe suggestion and generation, the project can be expanded to include features like nutritional analysis, meal planning, and real-time cooking assistance. This scalability enhances its utility and relevance in diverse contexts.

# 8. Impact on Health and Wellness: Encourages healthier eating habits by promoting homemade meals tailored to individual preferences and dietary needs. This aspect contributes to overall well-being and nutritional awareness among users.

# Overall, the "Customized AI Kitchen for India" project not only enhances culinary experiences but also promotes cultural appreciation, technological innovation, and community engagement around Indian cuisine.

# PROJECT SCOPE:

The scope of the "Customized AI Kitchen for India" project encompasses several key aspects that define its objectives, functionality, and potential applications. Here's an overview of its scope:

1. Recipe Suggestion and Generation: The project focuses on suggesting and generating recipes tailored to Indian cuisine preferences. It includes:

- Utilizing AI and natural language processing (NLP) models to recommend dishes based on user-specified cuisine types and available ingredients.

- Generating detailed recipe instructions for selected dishes, considering cultural cooking methods and ingredient combinations.

2. User Interaction and Personalization:

- Providing an intuitive interface for users to select preferred Indian cuisines and input available ingredients.

- Personalizing recipe suggestions to match dietary preferences, regional variations, and ingredient availability.

3. Educational and Cultural Exploration:

- Serving as an educational tool to introduce users to diverse Indian dishes, ingredients, and cooking techniques.

- Promoting cultural exploration by offering insights into regional specialties and traditional cooking practices.

4. Technological Integration:

- Integrating with advanced AI technologies like transformer-based models for efficient recipe recommendation and generation.

- Exploring potential integration with smart kitchen appliances or IoT devices for enhanced user experience and real-time cooking assistance.

5. Community Engagement and Collaboration:

- Facilitating community interaction through recipe sharing, user reviews, and collaborative content generation.

- Engaging users in discussions about food culture, culinary traditions, and innovative cooking methods.

6. Scalability and Future Development:

- Designing the project architecture to be scalable, accommodating future enhancements such as nutritional analysis, meal planning, and interactive cooking guides.

- Considering expansion into related domains like personalized meal delivery services or culinary education platforms.

7. Ethical Considerations:

- Addressing ethical implications such as data privacy, algorithm transparency, and cultural sensitivity in recipe recommendation and content generation.

- Ensuring inclusivity and accessibility for users with diverse dietary needs, preferences, and cultural backgrounds.

In summary, the scope of the "Customized AI Kitchen for India" project encompasses leveraging AI technologies to enhance culinary experiences, promote cultural appreciation, and facilitate personalized cooking solutions tailored to Indian cuisine enthusiasts. It aims to blend technological innovation with cultural preservation and community engagement in the realm of culinary art.

# TIME COMPLEXITY:

*To assess the time complexity of your code, we need to break down the primary operations within each function and understand their computational costs. Here’s a brief analysis*

*Function: `load\_model\_and\_tokenizer`*

*- Operations: Loading a tokenizer and model from the pre-trained models.*

*- Time Complexity:*

*- Loading the tokenizer and model involves fetching data from storage, which generally takes a constant time depending on the size of the models.*

*- Therefore, this can be considered \(O(1)\), but in practical terms, it’s dependent on I/O operations and model size.*

*Function: `suggest\_dish`*

*- Operations:*

*1. Tokenizing the input prompt: The complexity is \(O(n)\), where \(n\) is the length of the input prompt.*

*2. Generating the model output: The complexity depends on the model’s architecture. For a transformer model, this is typically \(O(m^2 \cdot d)\), where \(m\) is the sequence length and \(d\) is the model dimension.*

*3. Decoding the output: The complexity is \(O(k)\), where \(k\) is the length of the generated output.*

*4. Regex search and string operations: These operations are generally \(O(k)\) as they depend on the length of the generated output.*

*- Time Complexity: Combining these, the overall complexity can be considered \(O(n + m^2 \cdot d + k)\). Given that \(m\) and \(d\) can be substantial, the generation step \(O(m^2 \cdot d)\) is the dominant term.*

*Function: `generate\_recipe`*

*- Operations:*

*1. Tokenizing the recipe prompt: The complexity is \(O(n)\), where \(n\) is the length of the prompt.*

*2. Generating the model output: Similar to the `suggest\_dish` function, this is \(O(m^2 \cdot d)\).*

*3. Decoding the output: This is \(O(k)\), where \(k\) is the length of the generated recipe.*

*4. Regex search and string splitting: These operations depend on the length of the generated recipe, hence \(O(k)\).*

*- Time Complexity: The overall complexity can be considered \(O(n + m^2 \cdot d + k)\), with the generation step \(O(m^2 \cdot d)\) being the most significant.*

*Function: `main`*

*- Operations:*

*1. Listing cuisines and choosing one: This is a constant time operation, \(O(1)\).*

*2. Input handling: Generally \(O(1)\) for each input, but this is user-dependent and typically negligible in complexity analysis.*

*3. Calling `suggest\_dish` and `generate\_recipe`: These involve the complexities discussed above.*

*- Time Complexity: The overall complexity of `main` is dominated by the complexities of `suggest\_dish` and `generate\_recipe`, making it \(O(n + m^2 \cdot d + k)\).*

*Summary*

*- The dominant factor in the time complexity is the model’s generation step, which is \(O(m^2 \cdot d)\). This is due to the quadratic dependence on the sequence length \(m\) in transformer models.*

*- Other operations like tokenization, decoding, and string handling are linear in their respective input sizes.*

*Therefore, the most computationally expensive part of your code involves the transformer model’s generation process. The overall time complexity for suggesting a dish and generating a recipe can be summarized as \(O(n + m^2 \cdot d + k)\).*

# SPACE COMPLEXITY:

# To assess the space complexity of your code, we need to analyze the memory usage of each function and understand the primary contributors to the memory footprint. Here’s a brief analysis:

# Function: `load\_model\_and\_tokenizer`

# - Operations: Loading the tokenizer and model.

# - Space Complexity:

# - The space complexity is primarily determined by the size of the tokenizer and model.

# - Assuming the tokenizer and model are fixed in size, the space complexity can be considered \(O(1)\).

# - However, practically, the model size (\(M\)) and tokenizer size (\(T\)) are significant, so the actual space used is \(O(M + T)\).

# Function: `suggest\_dish`

# - Operations:

# 1. Tokenizing the input prompt: The space complexity is \(O(n)\), where \(n\) is the length of the input prompt.

# 2. Storing the input IDs: This also takes \(O(n)\) space.

# 3. Generating the model output: The memory required depends on the number of layers, sequence length \(m\), and model dimension \(d\). For transformer models, the space complexity is \(O(L \cdot m \cdot d)\), where \(L\) is the number of layers.

# 4. Storing the output tokens and decoded output: This takes \(O(k)\) space, where \(k\) is the length of the generated output.

# 5. Regex search and string operations: These operations use memory proportional to the length of the string being processed, hence \(O(k)\).

# - Space Complexity: Combining these, the overall space complexity is \(O(n + L \cdot m \cdot d + k)\).

# Function: `generate\_recipe`

# - Operations:

# 1. Tokenizing the recipe prompt: The space complexity is \(O(n)\), where \(n\) is the length of the prompt.

# 2. Storing the input IDs: This also takes \(O(n)\) space.

# 3. Generating the model output: Similar to `suggest\_dish`, this takes \(O(L \cdot m \cdot d)\) space.

# 4. Storing the output tokens and decoded output: This takes \(O(k)\) space.

# 5. Regex search and string splitting: These operations use memory proportional to the length of the string being processed, hence \(O(k)\).

# - Space Complexity: The overall space complexity is \(O(n + L \cdot m \cdot d + k)\).

# Function: `main`

# - Operations:

# 1. Listing cuisines and handling user input: These operations use constant space, \(O(1)\).

# 2. Input handling: Typically negligible in terms of space.

# 3. Calling `suggest\_dish` and `generate\_recipe`: The space complexity is dominated by these function calls.

# - Space Complexity: The overall space complexity of `main` is \(O(n + L \cdot m \cdot d + k)\), as it depends on the complexities of `suggest\_dish` and `generate\_recipe`.

# Summary

# - The dominant factor in the space complexity is the memory used by the transformer model, which is \(O(L \cdot m \cdot d)\). This is due to the multiple layers and large sequence length typically involved in such models.

# - Other operations like tokenization, decoding, and string handling are linear in their respective input sizes.

# Therefore, the most memory-intensive part of your code involves the transformer model’s generation process. The overall space complexity for suggesting a dish and generating a recipe can be summarized as \(O(n + L \cdot m \cdot d + k)\).

# SYSTEM DESIGN AND ARCHITECTURE:

# FUNCTIONAL REQUIREMENTS:

# IMPLEMENTATION DETAILS

# Certainly! Here are simplified functional requirements for your project:

# 1. User Input Handling:

# - The system should ask users to choose a cuisine type (like Indian, Italian) and enter available ingredients.

# 2. Recipe Suggestions:

# - Based on user inputs, the system should suggest unique recipes that match the selected cuisine and ingredients.

# 3. Recipe Generation:

# - When a user selects a specific recipe, the system should provide clear instructions on how to prepare that dish.

# 4. Output Display:

# - Recipes and instructions should be shown in an easy-to-read format, separating ingredients and steps clearly.

# 5. Error Management:

# - The system should handle mistakes gracefully, such as incorrect inputs, and guide users on how to correct them.

# 6. User Interface (UI):

# - The interface should be simple and intuitive, guiding users through each step clearly.

# 7. Performance:

# - The system should work efficiently, especially when generating recipes, to ensure quick responses.

# 8. Documentation:

# - Provide clear instructions on how to use the system and maintain it, including any setup requirements.

# These requirements outline what your project should do in a straightforward manner, focusing on user interaction, functionality, and usability. If you have any specific questions or need more details, feel free to ask!

# TOOLS AND TECHNOLOGIES USED:

# Certainly! Based on your project details, here are the key tools used and implementation details:

# Key Tools Used:

# 1. Google Colab:

# - \*Purpose\*: Used as the execution environment to run Python code, particularly beneficial for projects requiring computational resources like AI model inference.

# - Components:

# - Provides a Jupyter notebook interface hosted in the cloud.

# - Offers GPU and TPU acceleration options for faster model training and inference.

# 2. Transformers Library (Hugging Face)

# - Purpose: Essential for natural language processing (NLP) tasks using transformer-based models.

# - Components:

# - AutoTokenizer: Handles tokenization of text inputs into model-readable format.

# - AutoModelForSeq2SeqLM: Loads pretrained sequence-to-sequence models (e.g., T5) for generating text based on input prompts.

# 3. Python Programming Language:

# - \*Purpose\*: Core language for coding project logic, handling data processing, user interactions, and interfacing with external libraries.

# - Components:

# - Manages control flow, data structures (lists,dictionaries), and functions for modular code organization.

# Implementation Details:

# 1. Model Loading and Tokenization:

# - process: Loads the transformer model and tokenizer from Hugging Face's Transformers library.

# - Steps:

# - Initializes tokenizer to preprocess user inputs (cuisine type, ingredients) into tokenized sequences.

# - Loads pretrained model for generating recipe suggestions and detailed instructions based on tokenized inputs.

# 2. User Interaction and Input Handling:

# - Process: Engages users via Google Colab's interface to input preferences and available ingredients.

# - Steps:

# - Prompts users to select a cuisine type from a predefined list.

# - Collects a comma-separated list of ingredients available for cooking.

# 3. Recipe Suggestion and Generation:

# - Process: Utilizes loaded model to generate recipe suggestions and detailed instructions based on user inputs.

# - Steps:

# - Constructs input prompts combining selected cuisine type and ingredients for recipe generation.

# - Generates text sequences (recipe suggestions) using the transformer model and tokenization techniques.

# - Cleans and formats generated text for presentation to the user.

# 4. Output Formatting and Presentation:

# - Process :Ensures clear and structured presentation of recipe suggestions and instructions to the user.

# - steps:

# - Formats output to display suggested dishes with relevant details (title, ingredients, instructions).

# - Organizes recipe instructions into sequential steps, ensuring readability and coherence.

# - Displays output within Google Colab's notebook environment for user interaction and consumption.

# 5. Error Handling and Edge Cases:

# - Process: Implements error handling to manage unexpected user inputs or model-related issues.

# - Steps:

# - Validates user inputs for cuisine selection and ingredient lists to prevent processing errors.

# - Handles edge cases such as empty inputs or invalid selections gracefully with informative messages.

# 6. Integration and Expansion:

# - Process: Designs project architecture for scalability and potential integration with additional features or APIs.

# - Steps:

# - Optimizes model loading and memory management for efficient performance in Google Colab.

# - Plans for future enhancements like nutritional analysis, meal planning integration, or user feedback mechanisms.

# Summary:

# The implementation of your project leverages Google Colab for execution, the Transformers library for NLP tasks using transformer-based models, and Python for coding project logic and user interactions. This setup ensures efficient handling of recipe suggestion and generation tasks tailored to Indian cuisine preferences, showcasing technological integration and scalability in AI-driven applications. If you need further details or assistance with specific aspects, feel free to ask!

# CODE STRUCTURE OVERVIEW:

The code structure is organized into classes, functions, and modules to modularize and encapsulate different functionalities of the GPS Toll based System simulation:

* ***Classes:***
  + *Point: Represents a geographical point with latitude and longitude coordinates.*
  + *Polygon: Represents a polygonal shape defined by vertices (points). Used to define toll zones.*
* ***Functions:***
  + *move\_vehicle(vehicle, vehicle\_paths, steps): Simulates the movement of a vehicle along a predefined route.*
  + *check\_toll\_zone\_crossings(vehicle\_path, toll\_zones): Checks if a vehicle path intersects with any toll zone polygons.*

# SOURCE CODE: *Applies different toll rates (low, medium, high congestion) to simulate real-*

# !pip install transformers from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

# import re

# def load\_model\_and\_tokenizer():

# """Load the tokenizer and model."""

# print("Loading model and tokenizer...")

# tokenizer = AutoTokenizer.from\_pretrained("flax-community/t5-recipe-generation")

# model = AutoModelForSeq2SeqLM.from\_pretrained("flax-community/t5-recipe-generation")

# return tokenizer, model

# def suggest\_dish(tokenizer, model, cuisine\_type, ingredients):

# """Generate a dish suggestion based on the user's input."""

# input\_prompt = f"Suggest a {cuisine\_type} dish with {ingredients}"

# input\_ids = tokenizer(input\_prompt, return\_tensors="pt").input\_ids

# outputs = model.generate(input\_ids, max\_length=100, num\_return\_sequences=1, num\_beams=5, no\_repeat\_ngram\_size=2)

# # Decode output and extract the dish name

# decoded\_output = tokenizer.decode(outputs[0], skip\_special\_tokens=True).strip()

# match = re.search(r"^(.\*?)(ingredients:|directions:)", decoded\_output, re.IGNORECASE)

# if match:

# dish\_title = match.group(1).strip()

# else:

# dish\_title = decoded\_output.split(".")[0].strip() # Fallback to the first sentence

# return dish\_title

# def generate\_recipe(tokenizer, model, cuisine\_type, ingredients, selected\_dish):

# """Generate recipe instructions for the selected dish."""

# recipe\_prompt = f"Prepare a {cuisine\_type} dish with {ingredients}: {selected\_dish}"

# recipe\_input\_ids = tokenizer(recipe\_prompt, return\_tensors="pt").input\_ids

# recipe\_outputs = model.generate(recipe\_input\_ids, max\_length=300, num\_beams=5, no\_repeat\_ngram\_size=2)

# recipe\_instructions = tokenizer.decode(recipe\_outputs[0], skip\_special\_tokens=True)

# # Extract and clean up the instructions

# instructions\_match = re.search(r'directions?: (.\*)', recipe\_instructions, re.IGNORECASE | re.DOTALL)

# if instructions\_match:

# recipe\_instructions = instructions\_match.group(1).strip()

# else:

# recipe\_instructions = recipe\_instructions.strip()

# # Split instructions into steps

# recipe\_steps = re.split(r'\. |\n', recipe\_instructions)

# recipe\_steps = [step.strip() for step in recipe\_steps if step.strip()]

# return recipe\_steps

# def main():

# """Main function to run the recipe suggestion and display process."""

# tokenizer, model = load\_model\_and\_tokenizer()

# # List of available cuisines

# cuisines = [

# "Italian", "Mexican", "Chinese", "Indian",

# "French", "Japanese", "Thai", "Greek",

# "Spanish", "Korean"

# ]

# print("Please choose a cuisine type from the list below:")

# for idx, cuisine in enumerate(cuisines, start=1):

# print(f"{idx}. {cuisine}")

# # Ensure valid cuisine choice

# while True:

# try:

# cuisine\_choice = int(input("Enter the number of the cuisine you want a recipe for: ")) - 1

# if 0 <= cuisine\_choice < len(cuisines):

# cuisine\_type = cuisines[cuisine\_choice]

# break

# else:

# print("Please enter a valid number from the list.")

# except ValueError:

# print("Please enter a valid number.")

# ingredients = input("Enter the ingredients you have, separated by commas: ").strip()

# # Generate a dish suggestion

# suggested\_dish = suggest\_dish(tokenizer, model, cuisine\_type, ingredients)

# # Display the dish suggestion to the user

# print(f"\nHere is a dish suggestion based on your input: {suggested\_dish}")

# # Generate and display recipe instructions

# recipe\_steps = generate\_recipe(tokenizer, model, cuisine\_type, ingredients, suggested\_dish)

# print(f"\nInstructions to make {suggested\_dish}:")

# for step in recipe\_steps:

# print(step)

# if name == "main":

# main()

# RESULTS AND ANALYSIS

# USE CASES:

\Scenario: John wants to cook a traditional Indian dish using the ingredients he has at home.

1. \*User Action\*:

- John opens the AI Kitchen application in his browser.

- He selects "Indian" from the cuisine options and enters "rice, chicken" as his available ingredients.

2. \*System Response\*:

- The system processes John's inputs and suggests several unique Indian recipes that can be made with rice and chicken.

3. \*User Action\*:

- John reviews the suggested recipes and selects one titled "Crock Pot Chicken Tikka Masala".

4. \*System Response\*:

- The UI displays detailed cooking instructions for "Crock Pot Chicken Tikka Masala", including a list of ingredients and step-by-step preparation directions.

5. \*Outcome\*:

- John follows the provided instructions, cooks the dish, and enjoys a delicious homemade meal.

This use case demonstrates how the UI interacts with the user at each stage of the cooking process, guiding them from selecting a recipe to preparing it step-by-step. It highlights the intuitive nature of the interface and its ability to provide practical cooking assistance based on user preferences and available ingredients. If you have more specific aspects you'd like to explore or further details needed.