HARNESSING CHAT COMPLETIONS API AND JSON-MODE TO BUILD GENAI APPLICATIONS

1. INTRODUCTION

Talking with Large Language Models.

2. IMAGINE A HOUSE FINDING ASSISTANT.

Given a query like:

> *I want a house that has 2 bathrooms, located on a main road, have a guest room. It must have hot water heating*

2. IMAGINE A HOUSE FINDING ASSISTANT.

Get a response like

```
{
  "minBedrooms": 0,
  "bathrooms": 2,
  "mainroad": "yes",
  "guestroom": "yes",
  "hotwaterheating": "yes"
}
```

A house with a minimum price of KSH 50,000, without a guest room and has 2 bedrooms

Get a response like


```
{
  "minPrice": 50000,
  "bedrooms": 2,
  "bathrooms": 1,
  "guestroom": "no"
}
```

3. JSON, JSON,JSON...

JSON mode

As LLMs can generate text, they can also generate JSON.

- Varies according to model.
- OpenAI 4o intr

4. PROMPTING ?

The art and science of crafting effective inputs for language models

- Enhancing LLM performance on various tasks
- Developing effective interfaces between LLMs and users/tools
- Understanding and expanding LLM capabilities
- Improving LLM safety by controlling things to handle
- Augmenting LLMs with external knowledge(RAG)

5. LIMITATIONS OF LLMS

- Not being able to run code.
- Not being able to get access to real-time data.
- Not understanding what the user wants due to poorly typed prompts.

6. OVERVIEW OF CHAT COMPLETIONS API

An API style for interacting with LLMs

Designed & made popular by OpenAI

7. OVERVIEW OF CHAT COMPLETIONS API

- Enables conversational interactions with language models
- Takes a list of messages as input and returns a message

8. OVERVIEW OF CHAT COMPLETIONS API

```
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
  model="gpt-4o-mini",
  messages=[
    {"role": "system", "content": "You are a helpful assistant
    {"role": "user", "content": "Who won the world series in 2
    {"role": "assistant", "content": "The Los Angeles Dodgers
    {"role": "user", "content": "Where was it played?"}
  ]
)
```

KEY FEATURES OF CHAT COMPLETIONS API

- Supports for multi-turn conversations
- Allows for different message roles (system, user, assistant)
- Enables more natural and context-aware interactions
- To supply the history of the conversation/previous LLM outputs.

CHAT COMPLETION API MESSAGE TYPES

- System Messages
- User Messages
- Assistant Messages

SYSTEM MESSAGES

- Purpose: Set the overall behavior and context for the model
- Invisible to the end-user
- Persists throughout the conversation
- Sets rules, guidelines, and persona for the AI

Example:

```
[
  {
    "role": "system",
    "content": "You're a housing data assistant.
You will generate filters matching from a user's query in JSON
A house with a minimum price of KSH 50,000, without a guest ro
You'll return a JSON message with the format
{{
  "minPrice": 50000,
```

```
  "bedrooms": 2,
  "bathrooms": 1,
  "guestroom": "no"
}}
If a filter is not requested, set the field as null."
```

USER MESSAGES

- Purpose: Represent the human's input/query in the conversation
- Contain the actual queries or prompts
- Can include additional context or instructions

Example:

```
[
  {
    "role": "user",
    "content": "I want a house that has 2 bathrooms, located o
  }
]
```

ASSISTANT MESSAGES

-- Act as a form of short-term memory within a single API call

- Allow the model to reference and build upon previous responses
- Enable more coherent and context-aware multi-turn conversations

Example:

```
{
  "role": "assistant",
  "content": "Based on the sales data you provided, I can see
}
```

LANGCHAIN

a framework for developing applications powered by language models

- Prompt Engineering
- Memory Management
- LLM Configuration
- Output parsers

LANGCHAIN IMPLEMENTATIONS

- Python -> https://python.langchain.com/
- Javascript -> https://js.langchain.com/
- Flutter & Dart -> https://langchaindart.dev
- Golang -> https://tmc.github.io/langchaingo/
- Java -> https://docs.langchain4j.dev/

LANGCHAIN > PROMPT ENGINEERING

LangChain provides a simple and flexible way to define and customize prompts for language models.

1. Design your prompt
2. Specify the model to call.
3. Call the model with the prompt.
4. Handle the output.

DESIGN YOUR PROMPT

```
final prompt = ChatPromptTemplate.fromPromptMessages(
  [
    ChatMessagePromptTemplate.system("""
      You're a housing data assistant.
      You will generate filters matching from a user's query.
      If a filter is not requested, set the field as null. Ret
      Make sure you return valid JSON only.
      """
    ),
    ChatMessagePromptTemplate.human('{query}'),
  ],
);
```

SPECIFY YOUR LLM

```dart
final chatModel = ChatOpenAI(
  apiKey: '---',
  defaultOptions: const ChatOpenAIOptions(
    temperature: 1.6,
    model: 'gpt-4o-mini-2024-07-18',
  ),
);
```

TALK TO THE LLM

```
final chain = prompt.pipe(chatModel).pipe(JsonOutputParser());
final response = await chain.invoke({'query': '2 bedroomed hou
```

EXAMPLES RESPONSE 1

SYSTEM

You're a housing data assistant.
You will generate filters matching from a user's query.
If a filter is not requested, set the field as null. Return the filters in JSON only.
Make sure you return valid JSON only.

USER

Houses near a mainroad with 2 bedrooms

RESPONSE

Markdown Plain


```
{
  "filters": {
    "road": "mainroad",
    "bedrooms": 2
  }
}
```

EXAMPLES RESPONSE 2

SYSTEM

You're a housing data assistant.
You will generate filters matching from a user's query.
If a filter is not requested, set the field as null. Return the filters in JSON only.
Make sure you return valid JSON only.

USER

Houses near a mainroad with 2 bedrooms

RESPONSE

Markdown Plain


```
{
   "location": {
      "mainroad": true
   },
   "bedrooms": 2
}
```


(Note: Since you didn't specify any other filters, I've only included the location and bedrooms in the JSON response. If you want to add more filters, feel free to let me know!)

EXAMPLES RESPONSE 3

SYSTEM

You're a housing data assistant.
You will generate filters matching from a user's query.
If a filter is not requested, set the field as null. Return the filters in JSON only.
Make sure you return valid JSON only.

USER

Houses near a mainroad with 2 bedrooms

RESPONSE

Markdown Plain


```
{
"location": {
"distance_to_mainroad": "nearby"
},
"bedrooms": {
"min": 2,
"max": 2
}
}
```

CONSISTENCY

We need consistent output format that we can use in our programs. We need to let the LLM know what are the outputs we're expecting.

JSON SCHEMA

A JSON schema definition is a JSON object that defines the structure and data types of a JSON object, including the properties it should contain, their data types, and any constraints or validations that should be applied.

EXAMPLE JSON SCHEMA

```
{
  'type': 'object',
  'properties': {
    'setup': {
      'type': 'string',
      'description': 'The setup for the joke',
    },
    'punchline': {
      'type': 'string',
      'description': 'The punchline for the joke',
    },
  },
  'required': ['setup', 'punchline'],
}
```

OPENAI STRUCTURED OUTPUTS & JSON MODE.

A feature that allows the model to generate JSON output based on a given input and a JSON schema. Given a JSON schema, it will allow the model to consistenlty generate JSON that follows a specific schema. Enabling you to handle expected inputs & outputs.

HOUSING FILTERS EXAMPLE

```
{
  "type": "object",
  "properties": {
    "minPrice": {
      "type": ["integer", "null"]
    },
    "maxPrice": {
      "type": ["integer", "null"]
    },
    "minArea": {
      "type": ["integer", "null"]
    },
    "maxArea": {
      "type": ["integer", "null"]
    }
```

EXAMPLES 1 WITHOUT JSON SCHEMA ENFORCED.

A house with a minimum price of KSH 50,000, without a guest room thats near a mainroad

```
{
    "min_price": 50000,
    "near_main_road": true
}
```

EXAMPLE 2 WITHOUT JSON SCHEMA ENFORCED

I want a house that has 2 bathrooms, located on a main road, have a guest room. It must have hot water heating

```
{
    "bathrooms": 2,
    "location": "main road",
    "guest_room": true,
    "hot_water_heating": true
}
```

ENFORCING JSON SCHEMA WITH STRUCTURED OUTPUTS

```dart
final chatModel = ChatOpenAI(
  apiKey: '---',
  defaultOptions: ChatOpenAIOptions(
    temperature: 1.6,
    model: 'gpt-4o-mini-2024-07-18',
    responseFormat: ChatOpenAIResponseFormat.jsonSchema(
      ChatOpenAIJsonSchema(
        name: 'HouseFilters',
        description: 'Set of house filters',
        strict: true,
        schema: schema,
      ),
    ),
  ),
);
```

EXAMPLES WITH JSON SCHEMA ENFORCED

A house with a minimum price of KSH 50,000, without a guest room thats near a mainroad

```
{
  "bathrooms": 2,
  "mainroad": "yes",
  "guestroom": "yes",
  "hotwaterheating": "yes"
}
```

EXAMPLES WITH JSON SCHEMA ENFORCED

Show listings with a minimum price of 1000, a maximum price of 5000, at least 2 bedrooms, 2 bathrooms, and an area of at least 1000 square feet, located near a main road.

```
{
    "minPrice": 1000,
    "maxPrice": 5000,
    "minArea": 1000,
    "minBedrooms": 2,
    "bathrooms": 2,
    "mainroad": "yes"
}
```

EXAMPLES WITH JSON SCHEMA ENFORCED

Show listings with a minimum price of 8000, a maximum price of 15000, at least 4 bedrooms, and 3 bathrooms, and located in a preferred area, and with a basement.

```
{
    "minPrice": 8000,
    "maxPrice": 15000,
    "minBedrooms": 4,
    "bathrooms": 3,
    "basement": "yes",
    "prefarea": "yes"
}
```

Show listings with a minimum price of 3000, a maximum price of 6000, at least 2 bedrooms, 1 bathroom, and a minimum area of 800 square feet, and with either a guest room or a furnished room, and a maximum parking space of 3.A house with a minimum price of KSH 50,000, without a guest room thats near a mainroad

```
{
    "minPrice": 3000,
    "maxPrice": 6000,
    "minArea": 800,
    "minBedrooms": 2,
    "bathrooms": 1,
    "mainroad": "yes",
    "guestroom": "no",
    "maxParking": 3,
```

DEMO

6. CONCLUSION

- If interfacing with LLMs, just use JSON mode.
- If the model doesn't support JSON Mode, pass the JSON schema and threaten the LLM

Q&A

Questions from the audience?

KEY POINTS

- Importance of prompt engineering
- Power of Chat Completions API
- Crucial role of different message types
- Overcoming memory limitations with assistant messages