

DIT 637 Smart and Secure Systems

TT08A Experiencing LLM Chat

08/17/2024 Developed by Clark Ngo

08/17/2024 Reviewed by Sam Chung

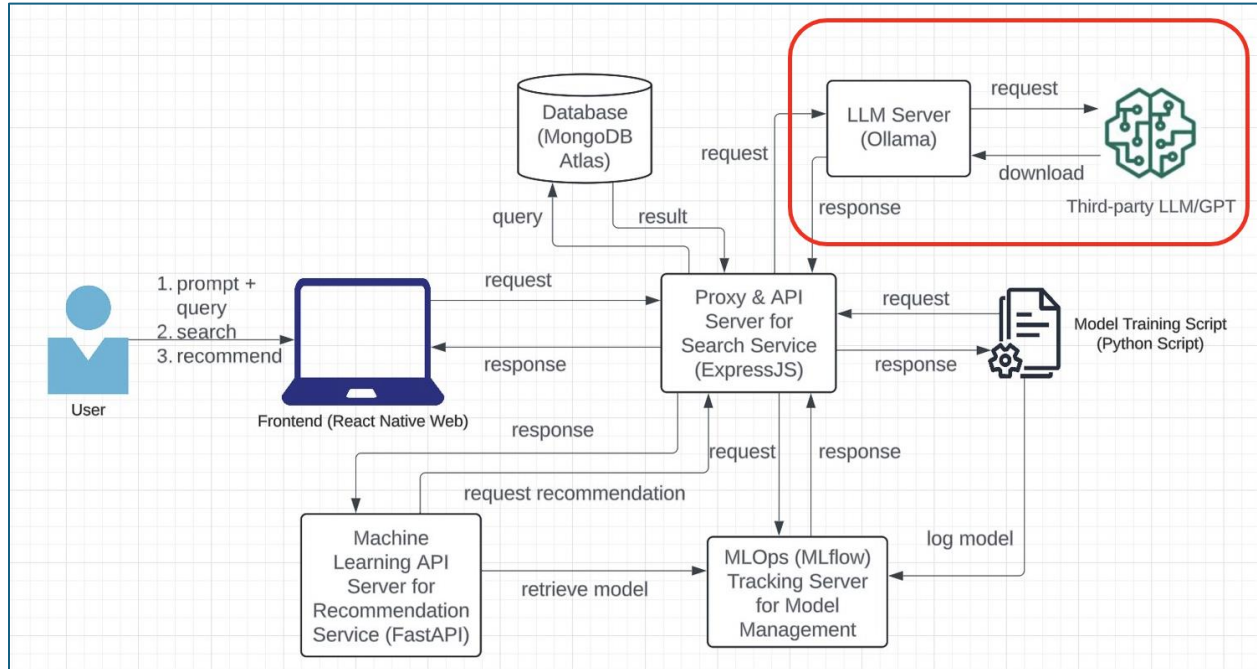
School of Technology & Computing (STC) @ City University of Seattle (CityU)



References

- Ollama. (n.d.). *GitHub - ollama/ollama: Get up and running with Llama 3.1, Mistral, Gemma 2, and other large language models*. GitHub. <https://github.com/ollama/ollama>
- Ollama. (n.d.). Ollama. <https://ollama.com/>
- GeeksforGeeks. (2024, June 3). *Ollama explained transforming AI accessibility and language processing*. GeeksforGeeks. <https://www.geeksforgeeks.org/ollama-explained-transforming-ai-accessibility-and-language-processing/>
- Banks, J. (2024, February 28). Gemma: Introducing new state-of-the-art open models. *Google*. <https://blog.google/technology/developers/gemma-open-models/>
- Team, G., & Deepmind, G. (n.d.). Gemma 2: Improving Open Language Models at a Practical Size. Retrieved August 18, 2024, from <https://storage.googleapis.com/deepmind-media/gemma/gemma-2-report.pdf>
- Google | Gemma 2 | Kaggle. (2024). Kaggle.com. <https://www.kaggle.com/models/google/gemma-2>
- *Getting started | Axios Docs*. (n.d.). <https://axios-http.com/docs/intro>

Key Concepts and Tools for Experiencing LLM Chat



LLM Server (Ollama)

Ollama, which Jeffrey Morgan and Michael Chiang developed, stands for Omni-layer Learning Language Acquisition Model. In the diagram, the **LLM Server (Ollama)** section handles requests related to the language model (LLM) operations, particularly for the chat interface in the frontend.

1. Frontend Interaction:

- The user interacts with the chat interface in the React Native web frontend, sending prompts or queries.
- These prompts are sent as requests to the **LLM Server (Ollama)** via the **Proxy & API Server (ExpressJS)**.

2. LLM Server (Ollama):

- The **LLM Server (Ollama)** receives and processes these requests using a third-party language model, such as a Generative Pre-trained Transformer (GPT).
- The **LLM Server** might either directly request the third-party LLM/GPT service or use a locally downloaded model to generate responses.

3. Streaming Response:

- The **LLM Server** is configured to send streaming responses back to the frontend. This means that as the LLM generates a response, it starts sending it back immediately rather than waiting for the entire response to be ready. It creates a smoother, real-time chat experience for the user.

4. Communication Flow:

- The processed response from the **LLM Server (Ollama)** is then sent back to the **Proxy & API Server (ExpressJS)**.
- The **ExpressJS** server forwards the response to the React Native frontend, which is displayed in the chat interface.

5. Third-party LLM/GPT:

- The third-party LLM/GPT (outlined in the red box) could be a service like OpenAI's GPT-3 or another similar model. It is the engine that generates the natural language responses based on the user's input.

Why Ollama?

Ollama is used for its advanced ability to understand and generate human-like text, enabling various applications from automation to educational support. Using advanced language models like those from OpenAI can streamline research by automating literature reviews, enhancing data analysis, and aiding in writing complex academic papers.

ExpressJS as Proxy Server and API Gateway

To streamline frontend requests to various services, we restructured our architecture so that ExpressJS serves as the sole service endpoint. ExpressJS will forward requests to different services, including the database, machine learning API server, MLOps tracking server, and LLM server. As with any system design, it is essential to consider and discuss the pros and cons of the architecture or design.

Pros:

- **Simplified Frontend:** A single endpoint (ExpressJS) reduces complexity in the frontend, making it easier to manage.
- **Centralized Routing:** ExpressJS handles routing, providing a single control point for directing traffic to the appropriate services.
- **Scalability:** The architecture can scale by managing different services behind the scenes, allowing independent scaling of backend components.
- **Security:** It is easier to enforce security measures like authentication and rate limiting at a single entry point.

Cons:

- **Single Point of Failure:** If ExpressJS fails, the entire system can be affected.
- **Increased Latency:** Routing requests through ExpressJS adds an extra layer, potentially increasing response times.
- **Bottleneck Risk:** High traffic could overwhelm ExpressJS, causing a bottleneck if not properly scaled.
- **Complexity in ExpressJS:** The service might become complex and more challenging to maintain as it takes on more responsibilities.

LLM with Gemma Model

Gemma is a family of lightweight, advanced open models developed by Google DeepMind, building on the research and technology behind the Gemini models. Named after the Latin word for "precious stone," Gemma models are designed for easy integration into applications, mobile devices, or hosted services. They can be customized using tuning techniques to excel at specific tasks. Supported by developer tools that encourage innovation and responsible AI use, Gemma models are intended for the AI development community to extend and enhance.

Gemma 2 and Gemma are key models in the Gemma family, designed for generative AI tasks like text generation, question answering, summarization, and reasoning. Built on the same research as the Gemini models, these base models come in various sizes, ranging from **2 billion to 27 billion parameters**, and can be tuned for specific use cases.

What are the parameters in the context of machine learning and deep learning?

Parameters refer to the components within a model learned from the training data. These parameters are essential for the model's ability to make predictions or generate outputs based on new inputs.

- **Weights and Biases:** In neural networks, parameters typically include weights and biases. Weights are the values that determine the strength of the connections between neurons in different layers of the network, while biases are additional constants added to the inputs of a layer to shift the activation function.
- **Training Process:** During training, the model adjusts these parameters to minimize the difference between its predictions and the actual target values (the loss). This process is typically done using optimization algorithms like gradient descent.
- **Model Size:** The number of parameters in a model often corresponds to its complexity and capacity to learn intricate patterns. For example, larger models with billions of parameters can capture more detailed relationships in the data, but they also require more computational resources and may need larger datasets to train effectively.

Inputs and outputs

Input: A text string, such as a question, a prompt, or a document to be summarized.

Output: Generated English-language text in response to the input, such as an answer to a question or a summary of a document.

Model Data

Data used for model training and how the data was processed.

Training Dataset

These models were trained on a dataset of text data that includes a wide variety of sources. The 27B model was trained with 13 trillion tokens, the 9B model was trained with 8 trillion tokens, and the 2B model was trained with 2 trillion tokens. Here are the key components:

- **Web Documents:** A diverse collection of web text ensures the model is exposed to a broad range of linguistic styles, topics, and vocabulary. Primarily English-language content.
- **Code:** Exposing the model to code helps it learn the syntax and patterns of programming languages, which improves its ability to generate code or understand code-related questions.
- **Mathematics:** Training on mathematical text helps the model learn logical reasoning and symbolic representation and address mathematical queries.

The combination of these diverse data sources is crucial for training a powerful language model that can handle a wide variety of different tasks and text formats.

Data Preprocessing

Here are the key data cleaning and filtering methods applied to the training data:

- **CSAM Filtering:** Rigorous CSAM (Child Sexual Abuse Material) filtering was applied at multiple stages in the data preparation process to ensure the exclusion of harmful and illegal content.
- **Sensitive Data Filtering:** As part of making Gemma pre-trained models safe and reliable, automated techniques were used to filter out certain personal information and other sensitive data from training sets.

- Additional methods: Filtering based on content quality and safety in line with our policies.

Benefits

At the time of release, this family of models provides high-performance, open large language model implementations designed from the ground up for Responsible AI development compared to similarly sized models.

Using the benchmark evaluation metrics described in this document, these models have shown to provide superior performance to other, comparably sized open model alternatives.

Uploading three image files to your GitHub Repository generated from GitHub Classroom

1. The screenshot of your 'OLLAMA LLM SERVER' as 'firstname_lastname_ollama_llm_server.png.'
2. The screenshot of your 'CHAT RECOMMEND' as 'firstname_lastname_chat_recommend.png.'

1) Use Case

As a movie enthusiast using a mobile device, I want to search and browse a list of movies with details such as title, genre, and year to quickly find information about movies I am interested in while on the go.

As a movie enthusiast using a mobile device, I want to get movie recommendations from my selections.

As a movie enthusiast using a mobile device, I want to chat and get movie recommendations from my selections.

Movie Recommender and Search Application

SEND MESSAGE

Hide

Initial Recommendations - Pick 3 Movies

Biography Napoleon Genres: Biography, Drama, History	Animation Winsor McCay, the Famous Cartoor Genres: Animation, Short, Comedy
Regeneration Genres: Biography, Crime, Drama	Gertie the Dinosaur Genres: Animation, Short, Comedy
Disraeli Genres: Biography, Drama, History	Steamboat Willie Genres: Animation, Family, Comedy
Salomè Genres: Biography, Drama, History	King of Jazz Genres: Animation, Music
Queen Christina Genres: Biography, Drama, History	Three Little Pigs Genres: Animation, Musical, Family

Submit Selected Movies

Search for Movies - Select any movie to get similar cast and genres

Load More Movies

Features included:

- Chat Recommendation
- Movie List Display
- Search Functionality
- Initial Recommendations
- New Recommendations
- Similar Cast and Genres

Movie Recommender and Search Application

SEND MESSAGE

Hide

Initial Recommendations - Pick 3 Movies

Biography Napoleon Genres: Biography, Drama, History	Animation Winsor McCay, the Famous Cartoor Genres: Animation, Short, Comedy
Regeneration Genres: Biography, Crime, Drama	Gertie the Dinosaur Genres: Animation, Short, Comedy
Disraeli Genres: Biography, Drama, History	Steamboat Willie Genres: Animation, Family, Comedy
Salomè Genres: Biography, Drama, History	King of Jazz Genres: Animation, Music
Queen Christina Genres: Biography, Drama, History	Three Little Pigs Genres: Animation, Musical, Family

Submit Selected Movies

Search for Movies - Select any movie to get similar cast and genres

Load More Movies

2) Setup

2.1) Running the Ollama LLM Server

- Create/Open GitHub Codespaces.
- Open a terminal and type the following:
 - **curl -fsSL https://ollama.com/install.sh | sh**

The command `curl -fsSL https://ollama.com/install.sh | sh` is used to download and execute a shell script from the specified URL. This specific script is designed to install Ollama on a Linux system. It detects the operating system architecture and installs the appropriate version of Ollama. The script also handles setup tasks, such as creating necessary directories and setting permissions.

 - `curl`: This is a command-line tool for transferring data with URLs.
 - `-f`: This option tells curl to fail silently when there are server errors.
 - `-s`: This makes curl operate silently without showing progress or error messages.
 - `-S`: When used with `-s`, it makes curl show an error message if it fails.
 - `-L`: This option tells curl to follow redirects.
 - `https://ollama.com/install.sh`: This is the URL of the script you want to download.
 - `| sh`: This pipes the downloaded script to the `sh` command, which executes it.
 - **ollama serve**

This command will start the server, and you can then interact with it using the Ollama API or other tools that communicate with the server.
- Test the Forwarded Address in the Browser. You can find the URL (port number: 11434) from the “PORTS” tab.



- Open a terminal and type the following:
 - **ollama pull gemma2:2b**
This command will download the Gemma 2 model with 2 billion parameters to your local environment, making it ready for use.
 - **ollama run gemma2:2b**
This command will start the Gemma 2 model, making it ready for inference tasks.

Gemma is a family of lightweight, state-of-the-art open models developed by Google. These models are designed for responsible AI development and are built from the same research and technology used to create the Gemini models

```
@clarkjasonngo → /workspaces/DIT637-TT08 (main) $ ollama run gemma2:2b
>>> limit 50 words. recommend me sci-fi movies.
**For mind-bending thrills:**

* **Inception (2010):** Dream sequences, elaborate heists, and mind-blowing twists.
* **The Matrix (1999):** Hacking reality and questioning the truth with spectacular action.

**For awe-inspiring adventures:**

* **Interstellar (2014):** Space exploration and human connection across galaxies.
* **Arrival (2016):** Communicating with alien life forms in a thrilling mystery.

Enjoy! 🚀

>>> Send a message (/? for help)
```

```
@samchung0117 → /workspaces/DIT637-TT08 (main) $ ollama run gemma2:2b
>>> recommend me 3 sci-fi. limit 25 words.
1. **Dune** by Frank Herbert: Political intrigue, giant sandworms, and a prophecy.
2. **The Martian** by Andy Weir: Solo astronaut fights for survival on Mars with ingenuity.
3. **A Fire Upon the Deep** by Vernor Vinge: Space opera at its finest, exploring the universe's future.

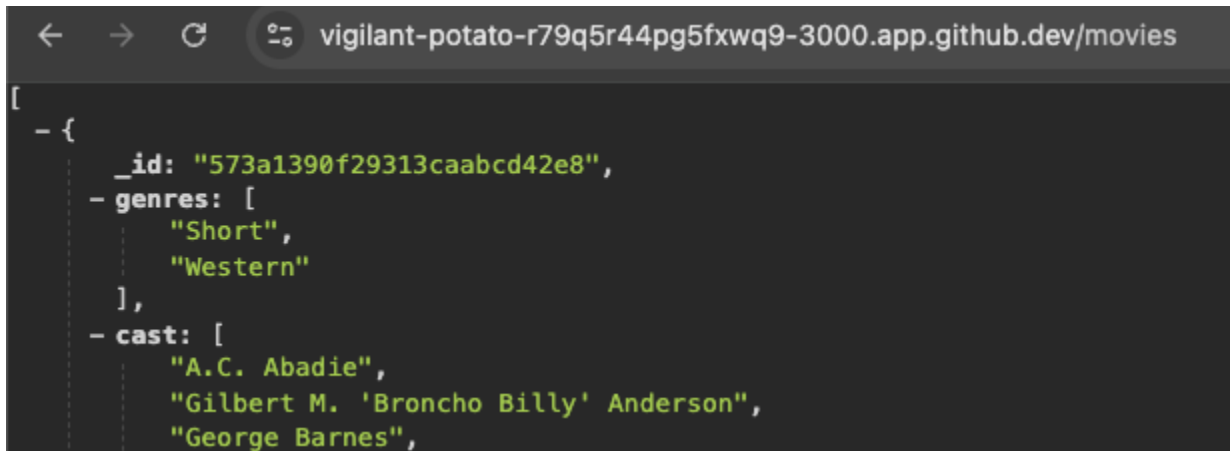
>>> Send a message (/? for help)
```

Take a screenshot of your 'OLLAMA LLM SERVER' as 'firstname_lastname_ollama_llm_server.png'.

- To exit the chat, press:
 - CTRL + D

2.2) Running the Backend ExpressJS Application Server

- Create a `.env` file inside 'backend-expressjs' folder (use example.env file as a reference)
 - Update the MONGODB_URI= connection string
- Open a new terminal:
 - `cd backend-expressjs`
 - `npm install`
 - `npm run start`
- Make the Port Public
- Copy the Forwarded Address (URL)
- Do a quick test if its running



The screenshot shows a web browser window with the address bar displaying `vigilant-potato-r79q5r44pg5fxwq9-3000.app.github.dev/movies`. The main content area shows a JSON response for a movie, with a dark background and light green text. The JSON structure is as follows:

```
[
  - {
    _id: "573a1390f29313caabcd42e8",
    - genres: [
      "Short",
      "Western"
    ],
    - cast: [
      "A.C. Abadie",
      "Gilbert M. 'Broncho Billy' Anderson",
      "George Barnes",
```

2.3) Running the MLflow Tracking Server

- Open a terminal and type the following:
 - **cd mlops_mlflow**
 - **pip install -r requirements.txt**
pip reads the “requirements.txt” file and installs all the listed packages with the specified versions
 - **mlflow server**
- Test the Forwarded Address in the Browser

```
@clarkjasonngo → /workspaces/DIT637-TT08/mlops-mlflow (main) $ mlflow server
[2024-08-17 17:41:56 +0000] [14512] [INFO] Starting gunicorn 22.0.0
[2024-08-17 17:41:56 +0000] [14512] [INFO] Listening at: http://127.0.0.1:5000 (14512)
[2024-08-17 17:41:56 +0000] [14512] [INFO] Using worker: sync
[2024-08-17 17:41:56 +0000] [14513] [INFO] Booting worker with pid: 14513
[2024-08-17 17:41:56 +0000] [14514] [INFO] Booting worker with pid: 14514
[2024-08-17 17:41:57 +0000] [14515] [INFO] Booting worker with pid: 14515
[2024-08-17 17:41:57 +0000] [14516] [INFO] Booting worker with pid: 14516
```

Access the MLflow Tracking Server

- Click the 127.0.0.1:5000 in the output (or access this in the Ports tab)

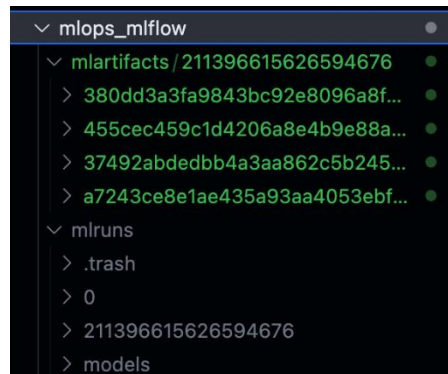
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 2		
Port	Forwarded Address	Running Process
● 5000	https://automatic-s...	/usr/local/python/3.10
● 11434	https://automatic-s...	ollama serve (8308)
Add Port		

The screenshot shows the MLflow 2.15.1 web interface. The top navigation bar includes the MLflow logo and tabs for Experiments and Models. The Experiments tab is active, displaying a search bar and a list of experiments. The 'Default' experiment is selected. Below the experiments list, there are tabs for Runs, Evaluation, Experimental, and Traces. The 'Runs' tab is active, showing a table of runs with columns for Run Name and Created. The table is currently empty.

2.4) Running the Python Scripts for ML Model Deployment (under the 'mlops-mlflow' directory)

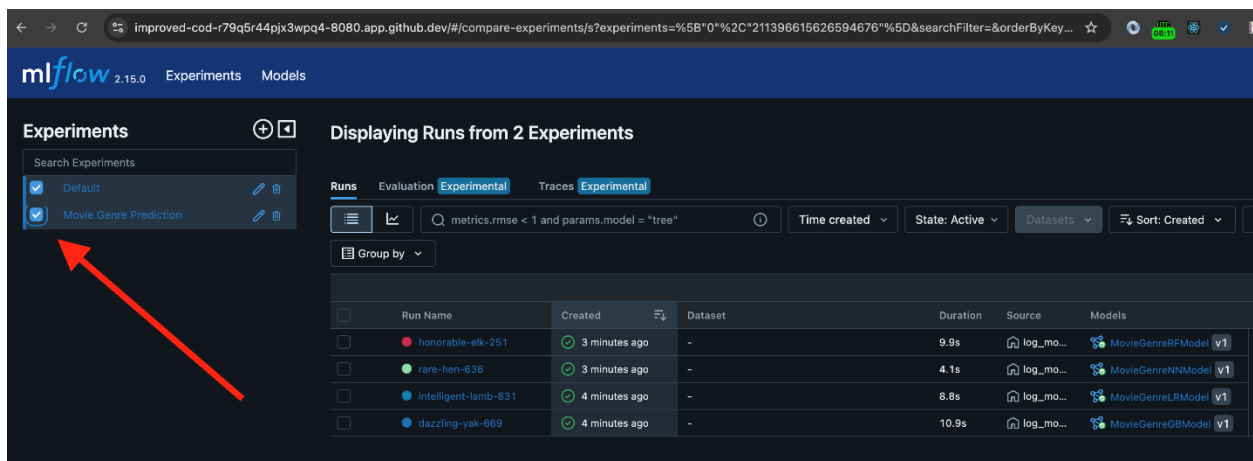
Train different model and prepare metadata for logging

- Open a new terminal:
 - `cd mlops-mlflow`
 - `python run_all_models.py`
- New folders 'mlartifacts' and 'mlruns' and their respective files will be generated and will be used as data in the MLflow Tracking Server.
 - **ML Artifacts:** Data files, models, and code created during machine learning experiments, stored for reproducibility, tracking, and reuse in future projects.
 - **ML Runs:** Individual executions of machine learning experiments, including configurations, parameters, and results, tracked to compare and improve models.



View the Experiments in the MLflow Tracking Server

- Head back to the server by clicking the Forwarded Address in the Ports tab.
- Tick the checkbox for Movie Genre Prediction to view all 4 models that were run.



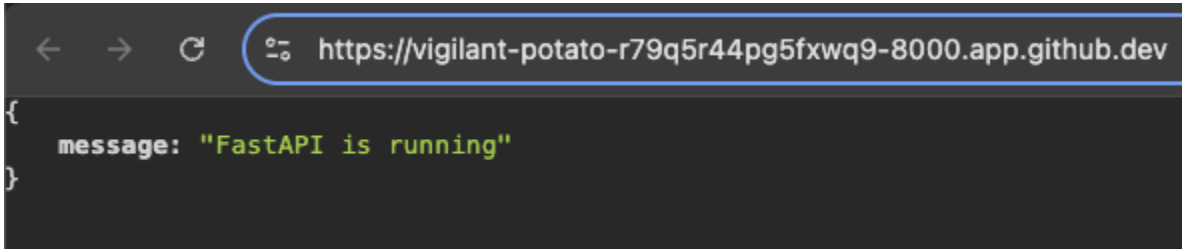
For example,

The screenshot displays the MLflow web interface for an experiment named 'Movie Genre Prediction'. The interface includes a sidebar with a search bar and a list of experiments, where 'Movie Genre Prediction' is selected. The main panel shows the experiment details, including tabs for 'Runs', 'Evaluation', 'Experimental', and 'Traces'. A search bar at the top of the runs section contains the query 'metrics.rmse < 1 and params.model = "tree"'. Below this, a table lists the experiment runs with columns for Run Name, Created, Dataset, Duration, Source, and Models.

Run Name	Created	Dataset	Duration	Source	Models
respected-fox-524	1 minute ago	-	9.6s	log_mov...	MovieGenreRFModel v1
nimble-shoat-787	1 minute ago	-	4.3s	log_mov...	MovieGenreNNModel v1
unruly-smelt-487	2 minutes ago	-	9.1s	log_mov...	MovieGenreLRModel v1
shivering-mule-364	2 minutes ago	-	10.4s	log_mov...	MovieGenreGBModel v1

2.5) Running the Machine Learning Model FastAPI Server

- Open a new terminal:
 - **cd modelserver-fastapi**
 - **pip install -r requirements.txt**
 - **uvicorn main:app --reload**
The 'uvicorn main:app --reload' is used to start a FastAPI application with Uvicorn, an ASGI (Asynchronous Server Gateway Interface) server.
- Copy the Forwarded Address (URL)
- Do a quick test if its running



2.6) Running the Frontend React Native Mobile/Web

- Create a **.env** file inside 'frontend-reactnative' folder (use the 'example.env' file as a reference)
 - Update the **API_URL= with ExpressJS Forwarded Address**
 - Please be mindful to NOT include a **"/" at the end**

```
frontend-reactnative > .env
1  # ExpressJS Forwarded Address
2  API_URL=https://automatic-space-lamp-r6g47v59gq925r5x-3000.app.github.dev
3
```

- Open a new terminal:
 - **cd frontend-reactnative**
 - **npm install**
 - **npx expo start --web** (or if mobile: **npx expo start --tunnel**)
- Open the Forwarded Address.

Movie Recommender and Search Application

I can definitely recommend some movies , but to give you the best s

Recommend me movies!

SEND MESSAGE

Hide

- Try to chat and send messages. For example:
 - "Recommend me 3 Romantic Movies."
- Take a screenshot of your 'CHAT RECOMMEND' as 'firstname_lastname_chat_recommend.png'.

3) Screenshot Summary

Chat: test the chat with any prompt. For example: "Recommend me 3 comedy movies".

Movie Recommender and Search Application

SEND MESSAGE

←

→

↺

🔍 automatic-space-la... ☆

📁

🎵

👤

Finish update ⋮

Movie Recommender and Search Application

1 . ** Dune ** - Epic space opera with political intrigue and giant sand worms .

SEND MESSAGE

4) Code Breakdown – Changes in ExpressJS as Proxy Server & API Server

FastAPI Communication Function

- **fetchFromFastAPI:** Sends POST requests to the FastAPI server and returns the response. Handles errors if the request fails.

```
const fetchFromFastAPI = async (endpoint, data) => {
  try {
    const response = await axios.post(`${FASTAPI_URL}${endpoint}`, data);
    return response.data;
  } catch (error) {
    console.error(`Error communicating with FastAPI: ${error}`);
    throw new Error('Failed to fetch from FastAPI');
  }
};
```

Axios is a *promise-based* HTTP Client for node.js and the browser. It is *isomorphic* (= it can run in the browser and nodejs with the same codebase). On the server-side it uses the native node.js http module, while on the client (browser) it uses XMLHttpRequests.

Routes

/chat (POST): Handles chat interactions, forwarding requests to the Ollama server and streaming the response back to the client in real-time.

```
app.post('/chat', async (req, res) => {
  try {
    let { model, messages } = req.body;

    // Set default model to 'gemma2:2b' if not provided
    model = model || 'gemma2:2b';

    // Set response headers for streaming
    res.setHeader('Content-Type', 'text/event-stream');
    res.setHeader('Cache-Control', 'no-cache');
    res.setHeader('Connection', 'keep-alive');

    // Forward the request to the Ollama server with streaming
    const response = await axios.post(`${OLLAMA_URL}/api/chat`, {
      model,
      messages,
    }, {
      responseType: 'stream'
    });

    // Pipe the streamed response data directly to the client
    response.data.on('data', (chunk) => {
      res.write(chunk);
    });

    // End the response when Ollama has finished sending data
    response.data.on('end', () => {
      res.end();
    });
  } catch (error) {
    console.error('Error communicating with Ollama:', error.message);
    res.status(500).json({ error: 'Failed to communicate with Ollama' });
  }
});
```

5) Code Breakdown – Frontend React Native Chat

Chat Function

The **handleChat** function is responsible for sending a chat request to a server and processing the streamed response in a React application. Here's a breakdown of what it does:

```
const handleChat = async () => {
  setIsLoading(true);

  try {
    const response = await fetch(`${API_URL}/chat`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        model: 'gemma2:2b',
        messages: [
          { role: 'user', content: userInput }
        ],
      }),
    });
  }
};
```

The code snippet you provided is an asynchronous function named `handleChat`. This function is designed to send a POST request to a server endpoint to handle chat messages. This function helps send chat messages to a server that processes them using the specified model (gemma2:2b).

Decode: The chunk of binary data (value) is decoded into a string.

```
if (response.ok) {
  const reader = response.body.getReader();
  const decoder = new TextDecoder('utf-8');

  while (true) {
    const { done, value } = await reader.read();
    if (done) break;

    // Decode the stream chunk
    const decodedChunk = decoder.decode(value, { stream: true });

    // Split the decoded chunk into lines
    const lines = decodedChunk.split('\n').filter(line => line.trim() !== '');

    for (const line of lines) {
      try {
        const parsed = JSON.parse(line);
        const content = parsed.message?.content?.trim();

        if (content) {
          setMessages((prevMessages) => [...prevMessages, content]);
        }
      } catch (error) {
        console.error('Error parsing JSON:', error);
      }
    }
  }
}
```

What is Streaming and Chunking?

Streaming delivers data in small, manageable pieces (chunks) as it's produced, rather than waiting for the entire data set.

Streaming, a process of continuous data transmission from a source to a destination, offers real-time benefits. It allows data to be processed and consumed as it arrives, making it particularly engaging for applications like video streaming, live broadcasts, and online gaming.

Chunking is a data transfer method in which the data stream is divided into smaller, manageable pieces called "chunks." Each chunk is preceded by its size in bytes, and the transmission ends when a zero-length chunk is received. This method is particularly useful when the content length is not known beforehand.

This allows immediate processing and display of information, reducing latency and memory usage, especially useful for large files or real-time data like video, audio, or continuous text responses.

The code creates a chat interface using React Native components.

- It includes a View container with a FlatList to display chat messages. The list is rendered horizontally, with a separator between items.
- A TextInput allows users to type their messages, and a Button sends the message when pressed.
- The FlatList uses index as a key for simplicity, and the Button is disabled while loading.

```
{/* Chat */}
<View style={styles.searchContainer}>
  <FlatList
    data={messages}
    renderItem={renderMessage}
    keyExtractor={({item, index}) => index.toString()} // Use index as key for simplicity
    horizontal // Enable horizontal scrolling
    ItemSeparatorComponent={renderSeparator} // Render space between items
    style={styles.messageList}
  />
  <TextInput
    style={styles.input}
    placeholder="Type your message here..."
    value={userInput}
    onChangeText={text => setUserInput(text)}
  />
  <Button title="Send Message" onPress={handleChat} disabled={isLoading} />
</View>
```

6) Pushing your work to GitHub

1. Go to Source Control on your GitHub codespaces and observe the pending changes.
2. Type the Message for your changes in the Message box on the top. For example, " **Submission for TT08 – Your Name**"
3. Click on the dropdown beside the commit button and select **Commit & Push** to update the changes to your repository main branch.
4. Select **Yes** when prompted.