

Real-time Large Language Model-driven AI Chatbots for Smart Tourism and Healthcare

Boxuan Yu, Glenda Yap, Roshini Pushparajan, Assoc Prof Chee Wei Tan

Introduction

- Generative AI is a pivotal aspect of AI, enabling models to autonomously create diverse content like text and videos.
- Recent advancements in Generative AI and Large Language Models (LLM) have indeed been remarkable.
- LLMs, such as OpenAI's GPT-3, are at the forefront of this revolutionary technology. These advanced systems are trained on massive datasets using sophisticated machine learning algorithms to understand human language patterns. Once internal structure has been developed, LLMs can generate humanlike responses to a wide range of user prompts.
- These empowering tools were initially available only for research but have now become more accessible to everyone, grabbing headlines' attention recently.
- This poster explores the practical capabilities of LLMs and uses real time data through API calls to make practical chatbots on the Nemobot platform.

Main features of our **Tour Guide Bot**

- Targets tourists and locals to have a seamless travelling experience, minimizing disruptions from weather or traffic conditions
- Features include :
 - Making use of [external API calls](#) from OpenWeatherMaps to obtain *real-time weather data*
 - Based on appropriate weather conditions, GetAttraction [LLM function](#) is called to randomly generate *tourist attraction suggestions*
 - Utilising [external API calls](#) from Singapore Tourism Board to get accurate *real-time bus arrival timing updates* to the local tourist spots

```
async function run(payload, state, tools) {
  const { isWeatherRequested, isAttractionRequested, isBusRequested, isBusTimingRequested } = state;
  const userResponse = payload.toString().trim();
  if (!isWeatherRequested && !isAttractionRequested && !isBusRequested) {
    state.isWeatherRequested = true;
    tools.reply("Would you like to get the current temperature and weather updates in Singapore?");
    return;
  }

  if (isWeatherRequested && !isAttractionRequested && !isBusRequested) {
    if (userResponse === "yes" || userResponse === "sure") {
      // API call to get real-time weather for Singapore
      const apiKey = "7c2c0f48484848484848484848484848";
      const apiUrl = "https://api.openweathermap.org/data/2.5/weather?lat=${state.latitude}&lon=${state.longitude}&appid=${apiKey}";

      try {
        // Fetch data from the API
        const response = await fetch(apiUrl);
        if (response.ok) {
          // Success: Network response was not ok
        } else {
          throw new Error("Network response was not ok");
        }
        const data = await response.json();
        console.log(data); // Log the weather data to the console

        // Extract the weather description and temperature from the data
        const weatherDescription = data.weather[0].description;
        const temperature = data.main.temp;

        // Reply to the user with the weather data
        tools.reply("The weather in Singapore is currently " + weatherDescription + " with a temperature of " + temperature + " degrees Celsius. Would you like attraction suggestions in Singapore? Please reply with 'yes' or 'no'." + state.isAttractionRequested = true;

      } catch (error) {
        console.error("Error fetching data:", error);
        tools.reply("Oops! Something went wrong while fetching the weather data.");
      }
    } else if (userResponse === "no") {
      tools.reply("Alright, enjoy!");
    } else {
      tools.reply("I didn't understand your response. Please reply with 'yes' or 'no'.");
    }
  }

  if (!isAttractionRequested && !isBusRequested && !isBusTimingRequested) {
    if (userResponse === "yes" || userResponse === "sure") {
      const GetAttraction = await tools.destructraction({ message: payload }); //attraction suggest
      tools.reply("Sure! Here are some attraction suggestions for you based on the weather condition: " + GetAttraction.suggestions);
      state.isAttractionRequested = true;
    } else if (userResponse === "no") {
      tools.reply("No problem, have a great time in Singapore!");
    } else {
      tools.reply("I didn't understand your response. Please reply with 'yes' or 'no'.");
    }
  }

  if (!isBusRequested) {
    if (userResponse === "yes" || userResponse === "sure") { //ask the user to enter the bus service
      tools.reply("Please enter the bus stop number and bus service number in the format 'busstop:busService'.");
      state.isBusRequested = true;
    }
  }
}
```

```
if (isWeatherRequested && !isAttractionRequested && !isBusRequested) {
  // Extract bus stop number and bus service number from user's message
  const [busStopNumber, busServiceNumber] = payload.split(":").map(part => part.trim());

  // Call STB API to get bus arrival timings
  const busArrivalResponse = await tools.callSTBAPI({ busStopNumber, busServiceNumber });

  // Process API response to generate response for chatbot
  const chatbotResponse = processBusArrivalResponse(busArrivalResponse);

  // Convert the chatbotResponse object to a JSON string with indentation
  const jsonString = JSON.stringify(chatbotResponse, null, 2);

  // Replace commas with commas followed by a line break character ("\\n")
  const modifiedString = jsonString.replace(/,/g, "\\n");

  // Send the modified string as a reply
  tools.reply(modifiedString);
  tools.reply("Enjoy your trip in Singapore!");
}

catch (error) {
  console.error("Error:", error);
  // Handle any errors and reply with an appropriate message
  tools.reply("Something went wrong. Please try again later.");
}
```

```
function processBusArrivalResponse(busArrivalResponse) {
  if (!busArrivalResponse.data || busArrivalResponse.data.length === 0) {
    return "Invalid bus stop number. Please input a valid number.";
  }

  const busData = busArrivalResponse.data[0];

  // Extracting relevant information
  const busServiceNumber = busData.serviceNumber;
  const nextBusArrival = new Date(busData.nextBus.EstimatedArrival);
  const nextBus2Arrival = new Date(busData.nextBus2.EstimatedArrival);
  const nextBus3Arrival = new Date(busData.nextBus3.EstimatedArrival);

  // Calculate arrival times in minutes
  const now = new Date();
  const nextBusTimeInMinutes = Math.round((nextBusArrival - now) / 60000);
  const nextBus2TimeInMinutes = Math.round((nextBus2Arrival - now) / 60000);
  const nextBus3TimeInMinutes = Math.round((nextBus3Arrival - now) / 60000);

  const responseMessage = {
    timings: `Bus number ${busServiceNumber} coming in (Minutes):`,
    nearestBus: nextBusTimeInMinutes,
    secondNearestBus: nextBus2TimeInMinutes,
    thirdNearestBus: nextBus3TimeInMinutes
  };

  return responseMessage;
}
```

```
if (!isBusTimingRequested) {
  try {
    // Extract bus stop number and bus service number from user's message
    const [busStopNumber, busServiceNumber] = payload.split(":").map(part => part.trim());

    // Call STB API to get bus arrival timings
    const busArrivalResponse = await tools.callSTBAPI({ busStopNumber, busServiceNumber });

    // Process API response to generate response for chatbot
    const chatbotResponse = processBusArrivalResponse(busArrivalResponse);

    // Convert the chatbotResponse object to a JSON string with indentation
    const jsonString = JSON.stringify(chatbotResponse, null, 2);

    // Replace commas with commas followed by a line break character ("\\n")
    const modifiedString = jsonString.replace(/,/g, "\\n");

    // Send the modified string as a reply
    tools.reply(modifiedString);
    tools.reply("Enjoy your trip in Singapore!");
  } catch (error) {
    console.error("Error:", error);
    // Handle any errors and reply with an appropriate message
    tools.reply("Something went wrong. Please try again later.");
  }
}
```

```
async function GetAttraction(userMessage) {
  // Give attraction suggestions based on the weather
  const apiKey = "7c2c0f48484848484848484848484848";
  const apiUrl = "https://api.openweathermap.org/data/2.5/weather?lat=${state.latitude}&lon=${state.longitude}&appid=${apiKey}";

  try {
    // Fetch data from the API
    const response = await fetch(apiUrl);
    if (response.ok) {
      // Success: Network response was not ok
    } else {
      throw new Error("Network response was not ok");
    }
    const data = await response.json();
    console.log(data); // Log the weather data to the console

    // Extract the weather description and temperature from the data
    const weatherDescription = data.weather[0].description;
    const temperature = data.main.temp;

    // Reply to the user with the weather data
    tools.reply("The weather in Singapore is currently " + weatherDescription + " with a temperature of " + temperature + " degrees Celsius. Would you like attraction suggestions in Singapore? Please reply with 'yes' or 'no'." + state.isAttractionRequested = true;

  } catch (error) {
    console.error("Error fetching data:", error);
    tools.reply("Oops! Something went wrong while fetching the weather data.");
  }
}

if (userResponse === "yes" || userResponse === "sure") {
  const GetAttraction = await tools.destructraction({ message: payload }); //attraction suggest
  tools.reply("Sure! Here are some attraction suggestions for you based on the weather condition: " + GetAttraction.suggestions);
  state.isAttractionRequested = true;
}

if (userResponse === "no") {
  tools.reply("No problem, have a great time in Singapore!");
}

if (userResponse === "I don't understand your response. Please reply with 'yes' or 'no'.") {
  tools.reply("I didn't understand your response. Please reply with 'yes' or 'no'.");
}
```

```
FUNCTION NAME
GetAttraction
DESCRIPTION
Give attraction suggestions based on the weather
PARAMETERS
userMessage
RETURNED TO CALLER
Parameters
userMessage
Return Value
A list of 5 random recommendations of Singapore attractions that are only separated by commas.
Example
["Recommendation 1", "Recommendation 2", "Recommendation 3", "Recommendation 4", "Recommendation 5"]
MESSAGE HISTORY
Include Memory (history messages from the chat)
```

```
user You ask to detect the weather from your previous message.
If rain or clouds or thunderstorm or drizzle, generate a list of 5 random recommendations of indoor Singapore attractions that are only separated by commas.
If sunny or clear sky, generate a list of 5 random recommendations of outdoor Singapore attractions that are only separated by commas.
This generates a list of 5 random recommendations of Singapore attractions that are only separated by commas.
Reply with the format below in JSON:
{
  "Recommendations": [
    "Recommendation 1",
    "Recommendation 2",
    "Recommendation 3",
    "Recommendation 4",
    "Recommendation 5"
  ]
}
assistant Message (Message)
Add Message
Add Tool
```

Extension : Main features of our **Medical Bot (MD)**

```
async function run(payload, state, tools) {
  const uuid = tools.uuid();
  await tools.chat_stream(
    (token) => {
      tools.streamMessage(uuid, token);
    },
    {userMessage: payload},
    {
      memory: tools.getChatHistory(100),
      externalTool: {
        "diagnose_symptoms": (userMessage) => {
          return userMessage;
        },
        "appointment_scheduling": (userMessage) => {
          return userMessage;
        },
        "medical_reminders": (userMessage) => {
          return userMessage;
        },
        "healthy_tips": (userMessage) => {
          return userMessage;
        }
      }
    }
  );
}
```

- User-friendly and caters to the everyday needs of public members to gain convenient access to important clinical information
- Harnesses capabilities of LLM functions through [external tools](#)
 - "*diagnose_symptoms*" tool enables users to communicate their symptoms and receive specialised diagnosis immediately.
 - "*appointment_scheduling*" with healthcare providers directly, streamlines process and optimises patient access to medical care.
 - "*medical_reminders*" tool encourages timely and precise medication intake, prioritising user's health outcomes.
 - "*healthy_tips*" tool provides users with practical advice and tips to maintain a healthy lifestyle
- With these powerful functionalities, MD empowers users to take control of their health, providing a dependable and convenient healthcare support system.

```
Temperature 0.7
Top P 0.95
Frequency Penalty 0
Presence Penalty 0
MESSAGE HISTORY
system You are a helpful assistant.
Your name is Nemobot.
Your reply should be concise, clear, informative and logical.
When you are asked about a question, you MUST think step-by-step. Explain each step in an ordered list.
Your reply should be in Markdown format.
Include Memory (history messages from the chat)
user {{userMessage}}
```

```
EXTERNAL TOOLS
Name
diagnose_symptoms
Description (A description for the bot, it will decide when to call a tool)
Prompt users to state medical symptoms. Identify symptoms in user message and predict a list of the 3 most possible medical conditions associated. State 1 piece of professional medical advice. Give 1 more piece of medical advice if prompted.
Parameter (JSON Schema)
{
  "type": "object",
  "properties": {
    "symptoms": {
      "type": "string",
      "description": "The user's symptoms"
    }
  },
  "required": ["symptoms"],
  "additionalProperties": false
}
```

```
async function yesOrNo(question, answer) {
  Detect if the answer is yes or no for the question.
  From this example, you will learn about how you can detect the user's intent from their message. Sometimes it's not as easy as passing the bot the question and the answer and let it guess, you also need to instruct it to think to improve the result.
  This is not the only prompt that works, please feel free to try other prompts and see if you can get better results.
  You can start with making it to only return yes or no to save some tokens.

  async function chat({userMessage, {memory, externalTool}}) {
    A chat bot that replies according to the previous conversation. It is also able to call 4 other functionalities through external tools.
  }
}
```

Attempts and takeaways

- As novices, navigating prompt engineering with LLM functions and combining JavaScript code for Nemobot's behavior was challenging.
- Missing out on reading Nemobot's OpenAI documentation initially led to many trial and errors, hence we sought advice from our mentors and ChatGPT for guidance.
- Lessons learnt :
 - Leveraging strengths of human-written code AND dynamic prompt engineering surpassed limitations of each approach in isolation (e.g. called API in code instead of in LLMs due to security risks)
 - Succeeded incorporating real-time data in the functions unlike static LLMs
 - Usage of precise instructions while conversing with Nemobot enabled us to utilise diverse facets of LLM functions(e.g. in-built human-like capabilities)

Future Plans: Medical Tourism

- Merging functionalities from both chatbots creates a brand-new, cutting-edge chatbot that effortlessly combines the worlds of travel and healthcare.
- Tailored for medical tourists, it offers a one-stop solution for individuals seeking high-quality medical treatments abroad while enjoying a seamless travel experience.
- From a comprehensive directory of accredited healthcare facilities, to real-time updates of the latest medical advancements, to multilingual support, these wide range of features ensure a smooth and informed medical tourism journey.

References

- [1] Chee-Wei Tan, Ching Nam Hang, Xintong Qi: A Chatbot-Server Framework for Scalable Machine Learning Education through Crowdsourced Data. ACM Learning at Scale 2022.
- [2] Chee Wei Tan, Shangxin Guo, Man Fai Wong, Ching Nam Hang, Copilot for Xcode: Exploring AI-Assisted Programming by Prompting Cloud-based Large Language Models, The International Joint Conference on Artificial Intelligence, Symposium on Large Language Models (LLM 2023).
- [3] Databricks. (2023). A Compact Guide to Large Language Models. <https://www.databricks.com/resources/ebook/tap-full-potential-llm>



NemoBot Platform QR



Tour Guide Bot QR



Medical Bot QR