Fine-Tuning a GPT Model for Smart Home Device Control

# Methodology & Approach

In this section, we outline the methodology and approach employed for fine-tuning a GPT model to interpret natural language commands for smart home device control. The process involves data preparation, model fine-tuning, function creation for device control, and model deployment for inference.

# Data Preparation

* The natural language commands, structured as a JSONL file, serve as the training data for the model.
* Initially, the data undergoes shuffling to ensure randomness and mitigate biases during training.
* A split of 80:20 is applied, where 80% of the commands are utilized for training, and the remaining 20% for evaluation.
* Shuffling the data enhances the robustness of the model by preventing it from learning patterns based on the order of the data.

def split\_file(input\_filename, train\_filename, eval\_filename, split\_ratio=0.8, max\_lines=100):

    """

    Splits the lines of the input file into training and evaluation files.

    """

    with open(input\_filename, 'r') as infile:

        lines = infile.readlines()

    # Shuffle lines to ensure randomness

    random.shuffle(lines)

    lines = lines[:max\_lines]

    # Calculate the number of lines for training

    train\_len = int(split\_ratio\*len(lines))

    # Split the lines

    train\_lines = lines[:train\_len]

    eval\_lines = lines[train\_len:]

    # Write to the respective files

    with open(train\_filename, 'w') as trainfile:

        trainfile.writelines(train\_lines)

    with open(eval\_filename, 'w') as evalfile:

        evalfile.writelines(eval\_lines)

    split\_file('./json\_conversations.jsonl', 'DATA\_train.jsonl', 'DATA\_eval.jsonl')

# Model Fine-Tuning

* The training and evaluation data files are uploaded to the OpenAI platform for fine-tuning the GPT model.
* The fine-tuning process is performed based on an existing model, such as **gpt-3.5-turbo**, to leverage pre-trained knowledge and expedite learning.

# Upload training data

train = openai.File.create(

    file=open('DATA\_train.jsonl', 'rb'),

    purpose='fine-tune'

)

# Retrieves the training model id

train\_id = train['id']

# Upload validation data

val = openai.File.create(

    file=open('DATA\_eval.jsonl'),

    purpose='fine-tune'

)

# Retrieves the validation model id

val\_id = val['id']

# Function Calling Feature

* Two functions, control\_device and set\_device\_mode, are designed to interpret natural language commands and transform them into actionable function calls.
* control\_device handles commands for turning on/off devices, while set\_device\_mode sets the mode of a device based on specific commands.

# --------------------------------------------------------------

# Use OpenAI’s Function Calling Feature

# --------------------------------------------------------------

function\_descriptions = [

    {

      "name": "control\_device",

      "description": "get the command to control a device",

      "parameters": {

          "type": "object",

          "properties": {

              "device": {

                  "type": "string",

                  "description": "device name"

              },

              "command": {

                  "type": "array",

                  "description": "A list containing the idx, type and val of the command",

                  "items": {

                      "type": "object",

                      "properties": {

                          "idx": {

                          "type": "string",

                          "description": "Represents the specific control within the device. For eg: L1 is turning on/off. L2 is after turning on then maybe increasing/decreasing the volume, temperature. L3 is changing channel so on. Each idx is an action to a specific device."

                          },

                          "type": {

                              "type": "string",

                              "enum": ["0x81", "0x80"],

                              "description": "Denotes the command type, such as 0x81 for on or start commands and 0x80 for off or stop commands."

                          },

                          "val": {

                              "type": "number",

                              "enum": [0, 1],

                              "description": ": Represents the state to which the device should be set, like 1 for on and 0 for off."

                          }

                        },

                        "required": ["idx", "type", "val"]

                    }

                }

          },

        "required": ["device", "command"]

      },

    },

    {

      "name": "set\_device\_mode",

      "description": "set the mode of a device",

      "parameters": {

          "type": "object",

          "properties": {

              "device": {

                  "type": "string",

                  "description": "device name"

              },

              "command": {

                  "type": "array",

                  "description": "A list containing the idx, type and val of the command",

                  "items": {

                      "type": "object",

                      "properties": {

                          "idx": {

                          "type": "string",

                          "description": "Represents the specific mode of the device, like L1 for mode 1, L2 for mode 2 and so on. Each idx can be"+

                          " of a different mode eg: low for mode 1, medium for mode 2, high for mode 3"

                          },

                          "type": {

                              "type": "string",

                              "enum": ["0x81", "0x80"],

                              "description": "Denotes the command type, such as 0x81 for on or start commands and 0x80 for off or stop commands."

                          },

                          "val": {

                              "type": "number",

                              "enum": [0, 1],

                              "description": ": Represents the state to which the device should be set, like 1 for on and 0 for off."

                          }

                        },

                        "required": ["idx", "type", "val"]

                    }

                }

          },

        "required": ["device", "command"]

      },

    }

]

# Model Deployment And Inference

* The fine-tuned model is deployed for inference using the Chat Completions API.
* Chat completions are generated based on user input messages, with function calling integrated to enable execution of device control actions.

completion = openai.ChatCompletion.create(

  model="ft:gpt-3.5-turbo-0125:personal::9NQll31i",

  messages=[

    {"role": "system", "content": "You are a smart home command operator that interprets natural language commands and transforms them into actionable function calls. Provide these details as a JSON dict."},

    {"role": "user", "content": "Turn on the kitchen light."}

    ],

    # Add function calling

    functions=function\_descriptions,

    function\_call="auto",  # specify the function call

)

# It automatically fills the arguments with correct info based on the prompt

# Note: the function does not exist yet

output = completion.choices[0].message

print(output)

By following this methodology and approach, we effectively fine-tuned a GPT model to interpret natural language commands for smart home device control, enabling seamless interaction between users and the smart home system.

# Performance Metrics And Validation Results

In this report, we present an analysis of the model's performance based on validation results obtained from comparing predicted actions with ground truth actions.

# Methodology

The validation process involved comparing the actions predicted by the model with the ground truth actions associated with each natural language command in the validation dataset. Accuracy was calculated as the proportion of correctly predicted actions out of the total number of actions in the dataset. The evaluation focused on understanding the discrepancies between predicted and ground truth actions to gain insights into the model's behavior.

# Results

The evaluation revealed significant discrepancies between the predicted actions generated by the model and the ground truth actions. While some parts of the commands were predicted accurately, particularly in cases where the commands were straightforward and unambiguous, the overall performance of the model varied across different command types.

For instance, most of the commands consisted of either controlling the device (turning on/off) or setting the device mode (setting temperature for eg.). In conclusion, it would hallucinate towards commands other than the types mentioned in the training file.

# Analysis

The discrepancies observed in the validation results indicate several areas of concern regarding the model's performance.

# Ambiguity in Commands

The model struggled to accurately interpret commands that were ambiguous or contained multiple possible interpretations. This led to inconsistencies in the predicted actions, as the model failed to discern the intended meaning of certain commands.

# Complex Command Structures

Commands with complex structures or multiple components posed challenges for the model, resulting in errors in predicting the sequence of actions. The model's ability to understand and execute multi-step commands was limited, leading to inaccuracies in the predicted actions.

# Training Data Quality

The discrepancies observed in the validation results may also be attributed to limitations in the quality and diversity of the training data. Inadequate representation of different command types and scenarios in the training dataset may have hindered the model's ability to generalize effectively to unseen data.

# Recommendations

To improve the performance of the model, the following recommendations are suggested:

# Data Augmentation

Augment the training dataset with additional examples covering a diverse range of command types and scenarios to improve the model's ability to generalize.

# Fine-Tuning Strategies

Explore alternative fine-tuning strategies, such as adjusting hyperparameters or using different pre-trained models, to enhance the model's learning capabilities.

# Conclusion

The validation results indicate that while the fine-tuned GPT model demonstrates some capability in interpreting natural language commands for smart home device control, significant challenges remain in achieving consistent and accurate performance. Addressing the identified issues, such as ambiguity in commands, improving the model's understanding of complex command structures, and enhancing domain-specific knowledge, will be crucial for enhancing the model's performance and usability in real-world applications.