Enabling Large Language Models as Intelligent Agents

Ajay Mittur
Language Technology Institute
Carnegie Mellon University
Pittsburgh, PA 15213
amittur@cs.cmu.edu

Ansh Khandelwal
Language Technology Institute
Carnegie Mellon University
Pittsburgh, PA 15213
anshk@cs.cmu.edu

Arushi Gupta
Language Technology Institute
Carnegie Mellon University
Pittsburgh, PA 15213
arushigu@cs.cmu.edu

Nihal John George
Language Technology Institute
Carnegie Mellon University
Pittsburgh, PA 15213
nihaljog@cs.cmu.edu

1 Abstract

This project investigates the use of large language models as intelligent agents capable of interacting with and performing tasks across diverse digital environments such as operating systems, databases, and virtual worlds. Developing such capable AI agents holds immense potential for revolutionizing human-computer interaction and automating complex digital tasks. We aim to reproduce the performance of state-of-the-art LLMs like Llama 2 on agent benchmarks and explore techniques to enhance their decision-making abilities through approaches like instruction fine-tuning and reasoning methods like Tree of Thought. We demonstrate significant gains in performance on tasks like database querying and operating system interactions.

2 Introduction

Intelligent agents that can interact with digital environments to accomplish complex goals have been a long-standing pursuit in artificial intelligence research. Traditional approaches, such as reinforcement learning (RL), have faced significant challenges when confronted with the vast state and action spaces inherent in diverse digital environments like operating systems, web browsers, and databases. The intractability of these domains has hindered the development of agents capable of performing tasks like retrieving stock prices, composing emails, or executing database queries at a satisfactory level.

Recent advancements in large language models (LLMs) have opened up promising avenues for addressing these challenges. LLMs, which are transformer-based models (and other architectures like state-space models) trained on internet-scale text sources, exhibit remarkable natural language understanding and generation capabilities. Their ability to complete paragraphs with highly probable sequences of text has enabled them to respond satisfactorily to human prompts, such as "Write an email to Alice about today's meeting."

Experiments with LLMs in domains like chess and generating functional GET requests have demonstrated their potential for interacting with digital environments. The "world model" and natural language abilities learned by these models during training seem to endow them with the versatility required to navigate the diverse landscapes of digital agents. This phenomenon has sparked interest in exploring the use of LLMs as agents capable of reproducing human-level performance in various digital environments.

The primary aim of this research is to reproduce the performance of an LLM agent in a given environment. By fixing a specific environment, we can investigate existing architectures and perform experiments in fine-tuning or vector representation-based approaches to augment the model's performance. Additionally, we aim to delve deeper into the reasons behind the success or failure of particular experiments and explore ways to interpret an LLM's results, enhancing our understanding of these models' decision-making processes.

The pursuit of LLM-based agents holds significant promise in revolutionizing the field of intelligent agents, enabling them to tackle complex tasks in diverse digital environments with human-like proficiency. By leveraging the remarkable capabilities of LLMs and advancing our understanding of their decision-making processes, we can unlock new frontiers in artificial intelligence and pave the way for more versatile and capable agents that can seamlessly interact with the digital world.

2.1 Problem Formulation

In this project, we aim to instruct the AI agent to perform the desired tasks.

Input: Training Data $\mathcal{D}_{\text{train}}$, agent system S, max training epoch E, early-stop threshold C
Output: Enhanced agent system $S_{\hat{\mathcal{F}}}$
Initialization: $i \leftarrow 0, r \leftarrow 0, H_0 \leftarrow \varnothing, \mathcal{F}_0 \leftarrow \varnothing.$
while $i < E$ **do**
  if $H_i \neq \varnothing$ **then**
    $\mathcal{F}_{i+1}$ = `AgentOptimizer.step`($\mathcal{F}_i$, H_i)
  else
    $\mathcal{F}_{i+1} \leftarrow \mathcal{F}_i$
  H_{i+1} = `Eval`($S_{\mathcal{F}_{i+1}}$, $\mathcal{D}_{\text{train}}$)
  if $H_{i+1}.loss < H_i.loss$ **then**
    H_i.fail_record $\leftarrow \emptyset$
    $\hat{\mathcal{F}} \leftarrow \mathcal{F}_{i+1}, i \leftarrow i+1, r \leftarrow 0$
  else
    H_i.failure_record $\leftarrow (\mathcal{F}_{i+1}, H_{i+1}.\text{loss})$
    $r \leftarrow r + 1$
    if $r > C$ **then**
      `// Early stop`
      `Break`
return $S_{\hat{\mathcal{F}}}$

Figure 1: Algorithm for Agent Training

The described algorithm outlines a systematic approach for training agent systems, utilizing iterative updates to enhance the function set based on historical performance. Beginning with initialization and setting parameters, it iterates through epochs, updating the function set using an optimizer and evaluating performance on training data. The algorithm employs roll-back mechanisms to revert updates that degrade performance and employs early-stopping criteria to halt training if performance deteriorates consistently. Ultimately, the best-performing function set is returned as the final trained agent system, ensuring optimal performance in real-world applications.

For us, the training and inference input data is purely text – interactions of an agent with an environment. This is explained in more detail in the dataset section

3 Literature Review

This literature survey embarks on a journey to elucidate the intricate process of using LLMs for effective deployment as agents in multifaceted environments. Our inquiry delves deeper into the nuanced methodologies employed to refine the performance of LLMs within these diverse environments. By scrutinizing previous approaches, we aim to unravel the intricacies of fine-tuning strategies, shedding light on the underlying mechanisms that drive the adaptability and efficacy of LLMs across various domains.

One line of research focuses on developing novel training strategies to equip LLMs with agent skills while preserving their general language abilities. Notably, [1] Zeng et al. (2023) introduced Agent-Tuning, a hybrid instruction-tuning approach that combines a lightweight dataset called AgentInstruct with general-domain instructions. AgentInstruct comprises 1,866 verified interaction trajectories across six diverse agent tasks, including household routines, online shopping, and web navigation. The authors employed a three-stage process involving instruction construction, trajectory interaction using GPT-4 as the agent, and trajectory filtering based on reward scores. To enhance the agent capabilities of LLMs while maintaining their general abilities, the authors fine-tuned the open-source Llama 2 series with a mixture of AgentInstruct and high-quality general-domain instructions, resulting in the AgentLM models (7B, 13B, and 70B parameters). Evaluations demonstrated that AgentLM significantly outperformed the original Llama 2 models on both held-in and held-out agent tasks, with the AgentLM-70B model approaching the performance of GPT-3.5 on unseen agent tasks. Importantly, AgentLM maintained its performance on general language tasks, such as knowledge, mathematics, and coding, demonstrating its ability to preserve general capabilities.

Another line of research focuses on refining language model decisions through adaptive feedback. [2] Wang et al. (2023) introduced AdaRefiner, which iteratively refines a language model's outputs by leveraging a verification model's feedback. AdaRefiner generates multiple candidate outputs, evaluates them using a verification model, and adaptively refines the outputs through a refinement model guided by the verification feedback. This feedback loop continues until convergence. AdaRefiner comprises three components: a base language model for initial outputs, a verification model for scoring output quality, and a refinement model for producing refined outputs based on feedback. The authors explored instantiations using a program verification model for mathematical reasoning and a natural language inference model for open-domain QA. Experiments showed AdaRefiner significantly improved performance over base language models on these tasks. It demonstrated strong generalization, outperforming prompting and constitutional AI on out-of-distribution tasks. The iterative refinement enabled correcting errors, handling ambiguities, and producing higher-quality, task-aligned outputs. This approach highlights using adaptive feedback to enhance language models' decision-making and reasoning abilities for complex tasks while mitigating errors and biases.

Another line of research aims to enable large language models to act as general zero-shot reasoners across diverse tasks through a novel instruction technique. [3] Ye et al. (2023) introduced the "Agent Instructs" framework, which involves providing the language model with a task instruction framed as a dialogue between an agent and a user. The key idea is to leverage the strong few-shot learning capabilities of large language models by providing an explicit instruction that guides the model's generation process. They found that intentionally framing instructions as dialogues between an agent and user better aligns with how large language models are pre-trained on dialogue-like data. This structured conditioning signal allows effectively adapting the model's general capabilities to new reasoning tasks. The authors first curated a dataset of 25,000 task instructions spanning 61 diverse reasoning tasks such as mathematical reasoning, common sense reasoning, and multi-step procedural reasoning. They then fine-tuned a large language model (LaMDA) on this dataset using a novel conditional generation objective that encourages the model to generate outputs consistent with the given task instruction. Extensive evaluations across the 61 tasks demonstrated that after fine-tuning with Agent Instructs, LaMDA achieved state-of-the-art zero-shot performance, outperforming both multi-task trained models and models that use more traditional prompting techniques. The approach proved effective across various reasoning modalities, including symbolic, multi-choice, and open-ended response formats. Notably, Agent Instructs enabled strong zero-shot generalization to unseen tasks and datasets, highlighting the framework's potential for building general-purpose reasoners. The authors also found that the generated agent dialogues provided insightful transparency into the model's reasoning process. This line of research showcases the power of leveraging large language models' few-shot learning capabilities through tailored instruction design for achieving general zero-shot reasoning abilities.

A different line of research explores self-supervised learning approaches to enable LLMs to leverage external tools and APIs. [4] tSchick et al. (2024) introduced Toolformer, a language model that learns to use external tools like search engines, calculators, and translation systems in a self-supervised manner. The model is fine-tuned on a large corpus augmented with sampled API calls that are filtered based on whether they reduce the perplexity of predicting future tokens. Toolformer can autonomously decide when and how to use different tools through simple API calls, requiring only a few examples for each API. The authors showed that Toolformer improved zero-shot performance on various downstream tasks, often outperforming much larger models. It can leverage tools such as question answering systems, calculators, Wikipedia search, machine translation, and calendars. Toolformer achieved substantial gains on tasks like fact completion, math problem solving, question answering,

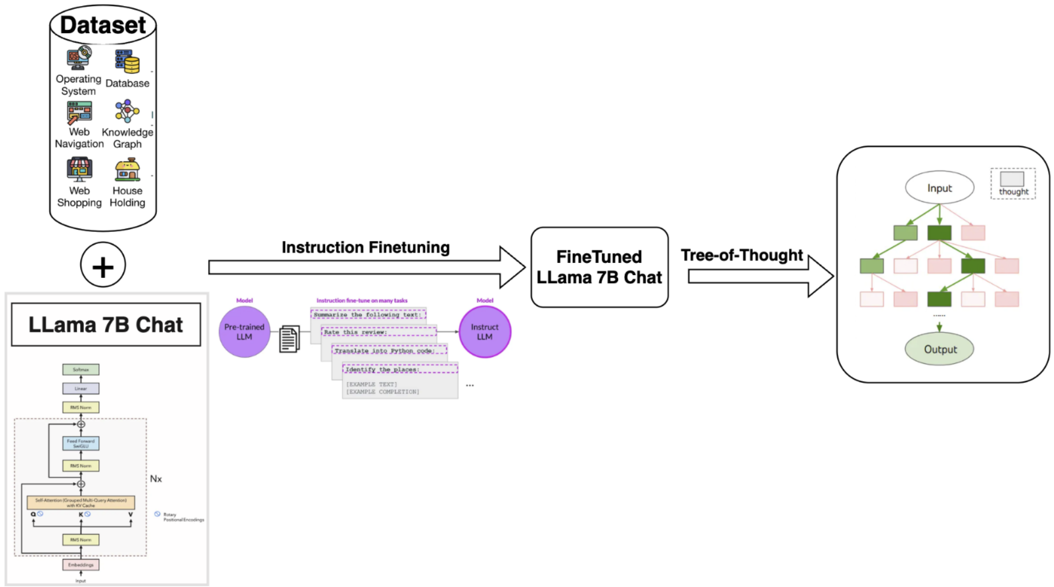

Figure 2: Model Description

and handling temporal information, without sacrificing core language modeling abilities. The paper demonstrated that language models can teach themselves to leverage external tools, combining their generalization capabilities with the precise information provided by specialized APIs.

These lines of research exemplify the diverse approaches being undertaken to enhance the agent capabilities of LLMs, ranging from novel training strategies to interpretability techniques and self-supervised learning for tool integration. As the field of AI agents continues to evolve, such research efforts hold promise for advancing the development of more capable, transparent, and adaptable AI systems capable of tackling complex real-world tasks.

4 Model Description

For this project, we aimed to leverage the capabilities of large language models as intelligent agents that can interact with and perform tasks in diverse digital environments like operating systems, databases, and virtual worlds. Our primary approach involved fine-tuning the Llama 7B Chat model, an open-source large language model from Meta specialized for open-domain conversational interactions.

We chose the Llama 7B Chat model as our base model for several reasons. Firstly, with 7 billion parameters, it has a substantial model capacity that allows it to capture and leverage rich knowledge from its pre-training on vast textual data. Additionally, its fine-tuning on a corpus of high-quality conversational exchanges endows it with strong language understanding and generation abilities, making it well-suited for the interactive and instruction-following nature of agent tasks.

To adapt the Llama 7B Chat model for our specific agent environments, we employed an instruction fine-tuning approach as mentioned in Figure 1. We curated a diverse dataset from AgentBench, a benchmark suite for evaluating large language models as agents across various tasks, including database querying, virtual world navigation, knowledge graph reasoning, operating system interactions, and card games. The details of this dataset are elaborated in the Dataset section.

The fine-tuning process involved training the model on this curated dataset using a technique called LoRA (Low-Rank Adaptation of Pretrained Models). LoRA is a parameter-efficient fine-tuning method that introduces a small number of trainable parameters to adapt the pretrained model to the target task, while keeping the original model weights frozen. This approach allows for efficient fine-tuning while preserving the model's general knowledge and capabilities. In our implementation, we utilized a LoRA rank of 8, which strikes a balance between model adaptability and computational efficiency.

To further enhance the model’s reasoning capabilities and decision-making processes, we employed the Tree of Thought (ToT) prompting technique. ToT encourages the language model to generate step-by-step reasoning chains, breaking down complex problems into intermediate steps. By explicitly reasoning through the problem and providing transparency into its thought process, the model can arrive at more accurate and coherent solutions, particularly for tasks that require multi-step reasoning or logical deductions.

Through this combination of instruction fine-tuning, model selection, and reasoning techniques, we aimed to create an intelligent agent capable of understanding and executing instructions within diverse digital environments, while exhibiting human-like reasoning and decision-making abilities.

5 Dataset

For the task of fine-tuning the Large Language Model (LLM) and subsequent testing, we opted to utilize the dataset provided by AgentBench, available at AgentBench GitHub repository. The dataset offers a comprehensive selection of tasks designed to evaluate the LLM’s performance across various domains. We further format the dataset to be ingestable by Llama 2 for finetuning by adding special tokens and converting dataset trajectories into readable instructions for the model. An example is shown below.

Sample Dataset Trajectory (Truncated for demo)

```
[{’from’: ’human’,
  ’loss’: None,
  ’value’: ’You are an assistant that will act like a person...’},
 {’from’: ’human’,
  ’loss’: None,
  ’value’: ’Now, I will start a new problem in a new OS. My problem is... directive.’},
 {’from’: ’gpt’,
  ’loss’: True,
  ’value’: ’Think: First, locate .conf files in the current directory....’},
 {’from’: ’human’,
  ’loss’: None,
  ’value’: ’The output of the OS:\n\nsample1.conf  sample2.conf\r\n\x1b]0;root@39909a690694...’},
 {’from’: ’gpt’,
  ’loss’: True,
  ’value’: ’Think: Then I need to concatenate the files and filter out the empty lines...’},
 {’from’: ’human’,
  ’loss’: None,
  ’value’: ’The output of the OS:\n\nparameter = value\r\ndirective = ...’},
 {’from’: ’gpt’,
  ’loss’: True,
  ’value’: ’Think: Now we need to process the lines to extract ...’}]
```

Converted to Llama 2 format (again truncated)

```
<s>[INST] <<SYS>>
You are a helpful, respectful and honest assistant.
<</SYS>>

You are an assistant that will act like a person,

Now, my problem is:

tell me how many files are in the directory "/etc"?bash:‘ [/INST] Think: To count the...

Act: bash
```

````
‘‘‘bash
ls /etc
‘‘‘</s><s>[INST] The output of the OS:
cpi cron.hourly fuse.conf iproute2 lvm networkd-dispatcher...[/INST] Think: ...

Act: bash
‘‘‘bash
ls -1 /etc | wc -l
‘‘‘</s><s>[INST] The output of the OS:
220 [/INST] Think: Now I get the answer, it is 220.

Act: answer(220)</s>
````

The special token `<s>` denotes the start of a user - agent chat sequence and `</s>` signifies the end. Everything between `[INST]` and `[/INST]` are instructions given to the model by the user. The text after it is the response we want the agent to learn. `<<SYS>>` is a special token that tells the model to consider everything following it until it encounters a `<</SYS>>` as a system prompt. A system prompt is something that the model always has to follow.

Dataset Split To facilitate the training procedure, we adopted a standard 75-25 split, allocating 75% of the data for training and reserving the remaining 25% for testing purposes.

Tasks for Fine-tuning and the data description We conducted fine-tuning on a shuffled dataset comprising instances from five distinct tasks, each requiring different skills and competencies from the LLM. These tasks are as follows:

1. **Database**: Involves querying and manipulating data within a relational database, necessitating proficiency in SQL commands and database operations. The database task dataset is constructed by amalgamating established datasets such as WikiSQL, WikiTableQuestions, SQA, HybridaQA, and FeTaQA, ensuring diverse instructions and data sources. Further enrichment is achieved through data augmentation using GPT-3.5-Turbo, resulting in 1599 entries categorized into select, insert, or update operations. Each sample includes an instruction, table info, table content, and correct answer, evaluated through an initialization, interaction, and checking process, with success rate measured across three categories: select, insert, and update operations.
2. **ALFWorld**: Challenges the agent to navigate and complete objectives within a virtual environment, testing its ability to understand and execute instructions in a simulated world. The ALFWorld dataset offers a comprehensive environment for evaluating Language Learning Models (LLMs) across various tasks, including navigation, object manipulation, and dialogue. It encompasses a diverse range of scenarios, from household chores to social interactions, each requiring nuanced language understanding and contextual reasoning. With detailed scene descriptions, interactive objects, and dialogue-based challenges, the dataset fosters LLMs' capabilities in real-world understanding and action execution.
3. **Knowledge Graph**: Requires the agent to query a knowledge base and utilize logic and reasoning to answer questions, assessing its capability to extract information from structured knowledge sources. The knowledge graph task dataset is meticulously curated from established KBQA datasets like GrailQA, ComplexWebQuestions, and GraphQuestions, focusing on long-term planning and decision-making abilities of LLMs. With 1,663 questions, each entry includes a natural language input question, topic entities, gold action sequence, and gold answer.
4. **Operating System**: Involves interacting with a simulated operating system environment, executing commands and scripts as instructed, evaluating the model's proficiency in shell scripting and system administration tasks. The OS dataset comprises 144 evaluation samples, featuring natural language instructions, docker environment configurations, and optional initialization and start scripts. Tasks include both Question Answering (QA) and Operation types, challenging LLMs to output commands for specific OS queries and verifiable operations. The dataset is curated from a mix of human-generated and GPT-4 generated queries, rigorously filtered for correctness and diversity.

5. **Card Game**: Tasks the agent with participating in a two-player card game, requiring strategic decision-making and gameplay tactics to emerge victorious, thereby assessing its ability to comprehend game rules and execute game-related actions effectively. The card game task dataset, based on the Aquawar framework, simulates interactive battles between two players controlling teams of pet fishes. With detailed game rules and player interactions, the dataset challenges models to strategically select actions and predict opponents' moves to ensure their team's survival.

6 Evaluation Metric

6.1 Evaluation Setup

For each problem (i.e., instruction), the execution can be divided into 3 parts [5].

1. Initialization. We create a docker container with a specific image, and we run an initialization bash script to set up environments specified by the instruction.
2. Interaction: A new shell is initiated within this docker environment, and the initial bash script specified by the instruction is run. Subsequently, the Large Language Model (LLM) under evaluation is provided with an instruction and the problem description. The interaction between the LLM and the shell begins, where in each turn, the LLM is presented with two actions: one to run a bash script, enabling the model to generate and execute a series of commands in the shell, and the other to commit an answer, terminating the interaction process. The model is deemed to have failed if it exceeds the round limit, typically set at 8 rounds.
3. Checking. For each problem, there is a checking pipeline containing a list of scripts $f_1, f_2, \cdots, f_n$, where f_k denotes the k-th script piece in the pipeline. For f_k, the answer of the model, o_0, and the output of $f_t(t < k)$, o_t, will be fed as input arguments into f_k, i.e., $o_k = f_k(o_0, o_1, \cdots, o_{k-1})$. The result is correct if and only if all the scripts exit with code 0.

6.2 Metrics

1. **Success Rate: [5]** the number of tasks successfully completed by the model divided by the total number of tasks
2. **Reward: [5]** This is basically equal to 0.7 x Winrate + 0.3 x Damagerate. Where, winrate is the number of rounds won out of total number of rounds. Damagerate is the total damage inflicted compared to total damage.

These evaluation metrics are carefully designed to assess the effectiveness and proficiency of the LLM agent in completing tasks within specific digital environments, aligning with the overarching objective of reproducing human-like performance in tasks such as OS and DB querying. By quantifying success rates and rewards, we aim to comprehensively evaluate the model's capabilities and its ability to interact seamlessly with the designated environments to accomplish complex goals.

7 Loss Function

The loss function serves as a critical metric for training our Large Language Model (LLM) agent to perform tasks within specific environments such as Operating Systems (OS) and Database (DB) querying. In our context, we employ **cross-entropy loss** to quantify the disparity between the model's predictions and the expected outcomes, thus guiding the training process towards improved task performance.

Let y_i represent the ground truth label for the i-th task, and $\hat{y}_i$ denote the predicted output of the model for the same task. Cross-entropy loss $\mathcal{L}_{CE}$ is computed as follows:

$$\mathcal{L}_{CE} = -\frac{1}{N} \sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

Where:

1. N is the total number of tasks.
2. y_i is the ground truth label for the i-th task.
3. $\hat{y}_i$ is the predicted probability assigned by the model to the i-th task being correctly performed.
4. $\mathcal{L}_{CE}$: Cross-entropy loss function.
5. N: Total number of tasks.
6. y_i: Ground truth label for the i-th task.
7. $\hat{y}_i$: Predicted probability assigned by the model to the i-th task being correctly performed.

The cross-entropy loss function plays a pivotal role in optimizing our LLM agent's parameters during training. By minimizing the cross-entropy loss, we aim to train the model to accurately predict the outcomes of tasks within specific environments, such as executing commands in operating systems or formulating correct database queries. This alignment ensures that the model learns to navigate and perform tasks effectively within these environments, ultimately striving towards the goal of reproducing human-like proficiency in OS and DB querying tasks using LLMs.

8 Baseline Selection

8.1 Baselines Models

We chose open source large language models coming from Meta as our baseline models. We choose the chat version of open Llama 2 (Llama-2-7,70b-chat) [6] (Touvron et al., 2023b) as our base models. Llama 2-7B Chat and Llama 2-70B Chat, are fine-tuned versions of the Llama 2 models specialized for open-domain conversation. The Llama 2-7B Chat is based on the 7 billion parameter Llama 2 model, while the Llama 2-70B Chat utilizes the larger 70 billion parameter version. Both models were fine-tuned on a vast corpus of web crawl data filtered for high-quality conversational exchanges, using a combination of supervised fine-tuning and unsupervised pretraining techniques. The fine-tuning process aims to adapt the models' language understanding and generation capabilities specifically for open-domain conversational interactions, enabling more natural and coherent responses. The Llama 2-70B Chat, with its increased model capacity, is intended for more advanced conversational tasks and applications, while the smaller Llama 2-7B Chat is designed for general open-domain conversational use cases. Both models leverage the Transformer architecture and are trained using a mix of supervised and unsupervised approaches.

Llama 2 models are particularly well-suited for agent AI tasks due to several key attributes. Firstly, they boast a broad knowledge base acquired through training on vast and diverse datasets comprising text from various sources such as the internet, books, and code repositories. This extensive knowledge equips them with a rich background that can be leveraged for tasks requiring general reasoning and understanding. Additionally, LLAMA 2 models excel in instruction following, as they are trained using techniques specifically geared towards understanding and executing natural language prompts and instructions. This flexibility makes them highly adaptable to new tasks specified through instructions, enabling seamless integration into diverse environments. Moreover, their multi-task capabilities are noteworthy, allowing them to perform well across a wide range of tasks including question answering, analysis, writing, and coding. This versatility is invaluable for creating AI agents that can handle a variety of tasks effectively. Furthermore, their strong language understanding capabilities enable them to interpret natural language instructions, descriptions, and feedback from humans accurately, facilitating smooth communication and interaction between agents and users. Finally, LLAMA 2 models possess the ability for few-shot learning, meaning they can adapt to entirely new tasks with minimal examples, thereby reducing the need for extensive task-specific data collection. These attributes collectively make LLAMA 2 models a compelling choice for agent AI tasks, offering a potent combination of knowledge, adaptability, versatility, and language understanding.

Table 2: Execution Outcomes of Failed Agents (Fraction of Runs)

Task	Llama 2 7B Chat				Llama 2 70B Chat			
	CLE	IF	IA	TLE	CLE	IF	IA	TLE
Database	0.0	0.0	1.0	0.0	0.0	0.89	0.0	0.023
ALFWorld	0.0	0.0	1.0	0.0				
Knowledge Graph	0.0	0.0	0.0	1.0				
Operating System	0.0	0.0	0.895	0.034	0.01	0.0	0.424	0.181
Card Game	0.0	0.0	1.0	0.0				

8.2 Baseline Performance

Table 1: Llama 2 Baselines on Tasks

Task	Llama 2 7B Chat	Llama 2 70B Chat	GPT-3.5-Turbo
Database	0.023	0.053	0.32
ALFWorld	0.0	_	0.13
Knowledge Graph	0.0	_	0.26
Operating System	0.014	0.055	0.37
Card Game	0.0	_	0.33

The ALFWorld task requires the agent to navigate a virtual environment, completing objectives guided by instructions. The Knowledge Graph task tests the agent's ability to query a knowledge base to answer questions using logic and reasoning. We evaluate an agent on this task using **Success Rate**. In the Operating System task, the agent interacts with a simulated operating system, executing commands as instructed. The **Success Rate** is used to evaluate this task. Further, in the Database task, the agent uses a relational database to query and manipulate data following precise instructions. The **Success Rate** is used to evaluate this task as well. The Card Game is a two-player battle with four fish cards per player. Each fish card has various attributes and abilities. The game consists of rounds where players take turns asserting the identity of an opponent's fish and then using a fish card's ability to attack an opponent's fish. The winner is the player with the most fish alive at the end of the game. The dataset includes details about the game mechanics, fish cards, and two baseline playing strategies. The evaluation process involves the LLM agent playing against a baseline agent and recording the results. Since this task is game based, the metric used is **Reward**.

9 Baseline Implementation

We followed a very similar methodology as the original paper to try and reproduce the baselines. We deployed Llama 2 7B chat and 70B chat models on AWS Sagemaker and implemented code to call the Llama 2 LLM on AWS for inference. The link to our implementation can be found on GitHub. The trajectories and prompts are formatted to Llama's prompt format with its special tokens, we pass the trajectories as a chat history alternating between user and agent and query the endpoint for inference. We also define the tasks we evaluate on in the AgentBench config files to tell the framework which tasks to run. Our results are shown in table 1. The results are comparable to the results mentioned in the original AgentInstruct paper [1]. We rule out hyperparameters or buggy implementation as the reason for baseline differences having tested our implementations and experimented with better generation parameters mentioned earlier extensively. There was no particular strategy to pick these parameters, we tried well known parameters that help an LLM generate coherent outputs such as top_p, top_k, max generation length, and the like.

On Llama-2-70b-chat we got better results than 7b, but again it falls short of the claimed performance by the paper on OS and DB. For other tasks – hosting 70b on AWS costed us $20 an hour. Due to insufficiency of AWS credits, the results for 70b now are only for OS and DB tasks.

To account for the differences we examine the reasons for failures and analyze the **Context Length Exceeded** (CLE) – interaction history exceeds LLMs context length, **Invalid Format** (IF) – the agent does not follow instruction format, **Invalid Action** (IA) – the agent chooses and invalid action, **Task Limit Exceeded** (TLE) – maximum interactions or rounds reached after agent fails to solve the task in the given time, and report the results in table 5.

We also investigate the discrepancy by looking into the agent trajectories, i.e. the conversation history with the LLM. For e.g. one of the IA trajectories for the Database task is shown below.

User: "I will ask you a question, then you should help me operate a MySQL database with SQL to answer the question.
You have to explain the problem and your solution to me and write down your thoughts.
After thinking and explaining thoroughly, every round you can choose to operate or to answer.
your operation should be like this:
Action: Operation
‘‘‘sql
SELECT * FROM table WHERE condition;
‘‘‘
You MUST put SQL in markdown format without any other comments. Your SQL should be in one line.
Every time you can only execute one SQL statement. I will only execute the statement in the first SQL code block. Every time you write a SQL, I will execute it for you and give you the output.
If you are done operating, and you want to commit your final answer, then write down:
Action: Answer
Final Answer: [ÄNSWER1, ÄNSWER2, ...]
DO NOT write this pattern unless you are sure about your answer. I expect an accurate and correct answer.
Your answer should be accurate. Your answer must be exactly the same as the correct answer.
If the question is about modifying the database, then after done operation, your answer field can be anything.
If your response cannot match any pattern I mentioned earlier, you will be judged as FAIL immediately.
Your input will be raw MySQL response, you have to deal with it by yourself.

Agent: Ok

User: Set the score of tie number 1 in the Football Matches table to ’2 - 2’.
The name of this table is Football Matches, and the headers of this table are Tie no,Home team,Score,Away team,Date,Attendance.

Agent: Sure! Here’s the SQL query to set the score of tie number 1 in the ‘Football Matches‘ table to ’2 - 2’:
‘‘‘sql
UPDATE FootballMatches
SET Score = ’2 - 2’
WHERE TieNo = 1;
‘‘‘
This query will update the ‘Score‘ column for the tie with ‘TieNo‘ equal to 1 in the ‘Football Matches‘ table.

As you can see, the agent is able to generate correct looking outputs and not do too bad, however, it is not able to follow instructions properly. Its mistake in the above trajectory was that it did not specify the sql command it wanted to run after "*Action: Operation*", instead it listed it out directly. So, 7B Chat does not do too well in agent environments where it needs to adhere to a specific format of outputs. This is the same pattern we observe the agent do in other tasks as well where its performance is low.

10 Implemented Extensions

Inspired by AgentTuning, we follow a similar approach of fine tuning Llama 2 7B Chat on an instruction set consisting of interaction trajectories in each of the environments. Further, we also implement a greedy tree-of-thought reasoning methodology with a branching factor of 3 to explore if it improves model performance on the tasks as it does for large LLMs. We found that the adapter rank of 16 gave the best trend. Fig. 3 shows some of the ablations we tried and hence these plots are for finetuning 8-bit quantized model weights.

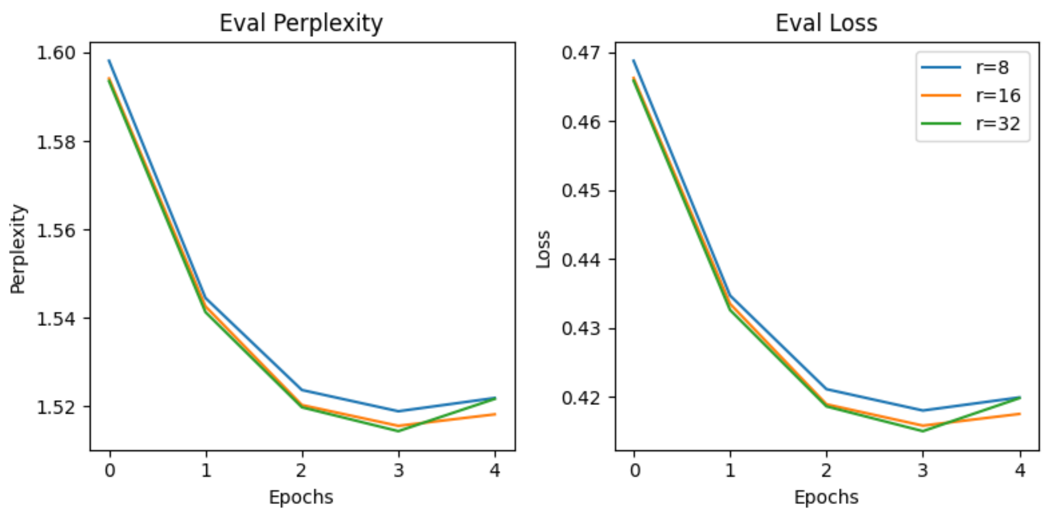

Figure 3: Evaluation Metrics for Different Ranks

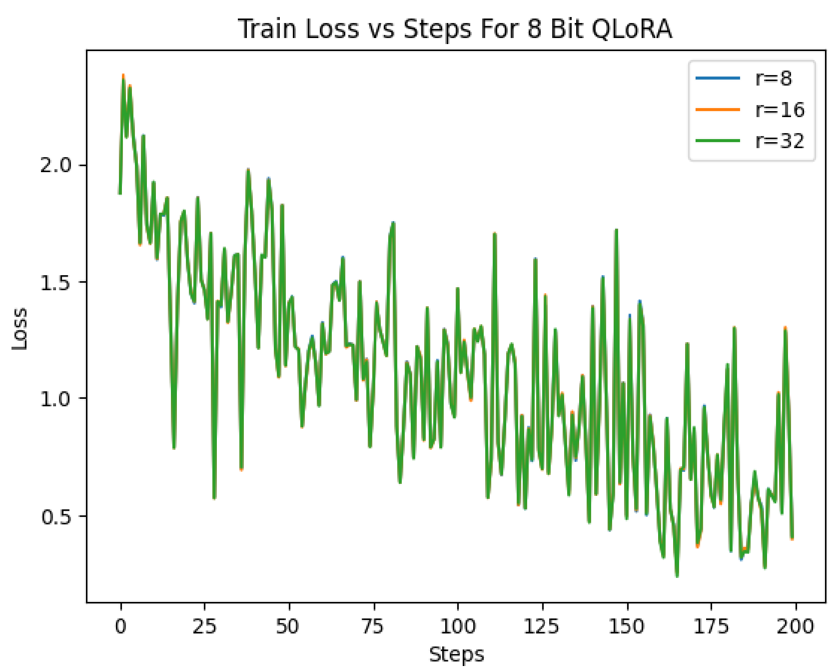

Figure 4: Effect of LoRA Ranks on Loss

All of the three ranks follow the same trend when it comes to training loss. There wasn't any significant difference as seen in Fig. 4. It's very noisy because of our very small batch size of 1.

To ensure we do not exceed our budget, we found the best hyperparameters over 5 epochs on a quantized version of Llama 2 7B Chat and then used those hyperparameters to perform full precision finetuning over 13 epochs. 13 epochs was took us around 26 hours and was the max we could finetune on `ml.g5.12xlarge`, which costs $7/hr, before exceeding out budget. This is also the reason we stuck to batch size 1 (the next greater suitable compute is `ml.g5.24xlarge` which is double the cost). We list our final sagemaker hyperparameters in table 3.

Hyperparameter	Value
epoch	13
enable_fsdp	True
int8_quantization	False
learning_rate	0.0001
lora_alpha	32
lora_r	16
lora_dropout	0.05
max_input_length	1024
batch_size	1

Table 3: Hyperparameters Used to Fine-tune Llama 2 7B Chat

In Fig. 5, we plot the perplexity and cross entropy loss of the model as it is fine tuned over 13 epochs using LoRA []. We perform cold restarts at epochs 4 and 9 to make sure we don't overfit to the training set. We observe a good downward trend in both the perplexity and loss.

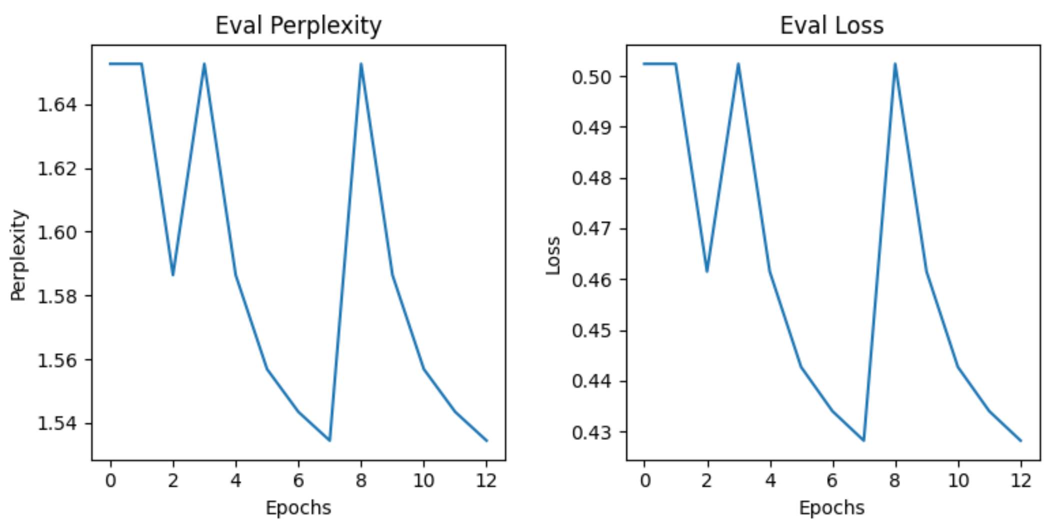

Figure 5: Instruction Tuning

Later we implement our own tree-of-thought using a greedy search strategy over the action space with a branching factor of 3. It consists of a rollout phase where the agent proposes next actions to take and a critic that evaluates how good an action is. Both of these phases are implemented on the same Llama model since it now has a better reasoning capabilities over the tasks. Then based on a score assigned to the actions (1 to 3, with 3 being a good action), the agent selects the best action to perform. Note that we also pass the entire interaction history of the agent so that the actions it proposes and evaluates are a posterior over the trajectory allowing it find better actions.

11 Results

We compare the results of our agent with the baselines in table 4. Our instruction tuned Llama 2 7B Chat performs significantly better than the baseline and even the base 70B Chat variantin all the tasks except ALFWorld and Knowledge Graph. However, we see that tree-of-thought performs worse than all the models with it being unable to follow instructions and even output valid actions in all environments except Operating System. Table 5 shows the various types of outcomes of our instruction tuned model compared to the base Llama 2 7B Chat LLM. We did not investigate this for tree-of-thought since it was so bad, that there would be no point. Even changing the branching factor did not make any difference. We discuss our results in detail in section 12.

Table 4: Performance of Our Approach

Task	Instruct Tuned	ToT	7B Chat	70B Chat	GPT-3.5-Turbo
Database	0.27	-[2]	0.023	0.053	0.32
ALFWorld	0.0	-[2]	0.0	-[1]	0.13
Knowledge Graph	0.0	-[2]	0.0	-[1]	0.26
Operating System	0.11	0.002	0.014	0.055	0.37
Card Game	0.003	-[2]	0.0	-[1]	0.33

Table 5: Execution Outcomes of Failed Agents (Fraction of Runs)

Task	Instruct Tuned				Llama 2 7B Chat			
	CLE	IF	IA	TLE	CLE	IF	IA	TLE
Database	0.0	0.43	0.0	0.02	0.0	0.0	1.0	0.0
ALFWorld	0.0	0.0	0.26	0.74	0.0	0.0	1.0	0.0
Knowledge Graph	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
Operating System	0.0	0.0	0.27	0.16	0.0	0.0	0.895	0.034
Card Game	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0

12 Discussion

As shown in the results, our instruction tuned variant performed significantly better than the base and 70B variant of Llama 2. To illustrate a successful environment interaction, we visualize our agent's behavior in Fig. 6. The task is to find some piece of information by querying SQL databases and you can see how the model does that successfully. Note that we have truncated the full dialogues to make the image more readable.

[2] Agent performs invalid action and errors out

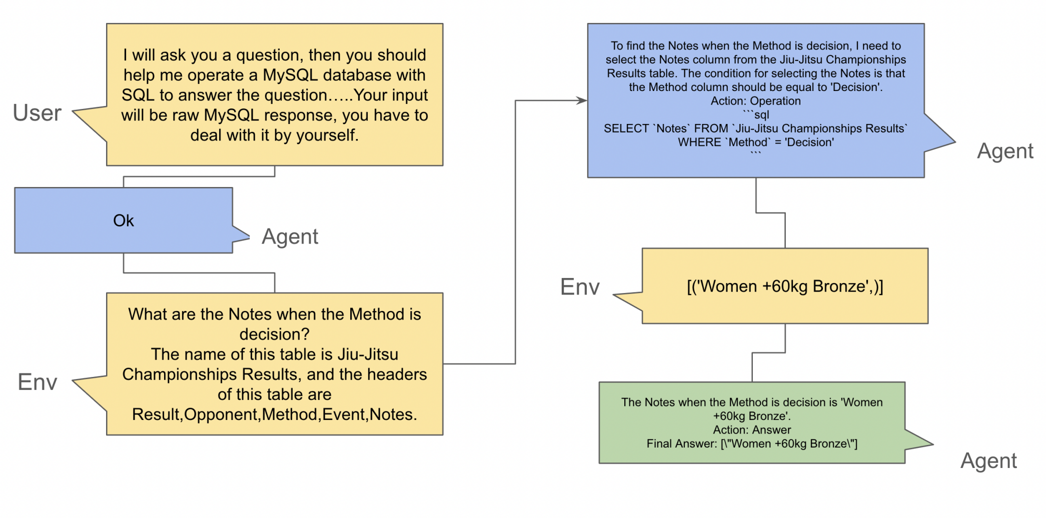

Figure 6: Instruct Tuned Success on DBBench

This is just one of the improvements we see in our agent after improving its agentic capabilities by instruction tuning over similar trajectories. We investigate the different failures of our agent as well so as to cover all bases. Table 5 also shows several improvements in cases where the agent is not able to succesfully complete a task as well. Notably we see drastic reductions in the percentage of invalid actions (IA) taken by the agent and an increase in task limit exceeded (TLE). While this may not sound good, it is encouraging because we know that the agent is able to reason and form more thoughts instead of taking an invalid action. The agent has gone from taking invalid actions to now thinking more about what it needs to do to achieve a task. It is also to be kept in mind that this was just after 13 epochs. If we had the budget to train the model further, we anticipate that we could have achieved even larger gains.

Our investigation into Tree-of-Thought (ToT) for these tasks is novel. While no published research at top ML conferences has established a ToT baseline on this dataset (released last year), our initial experiments with a 70B parameter LLM (Llama) suggest that effectiveness might be limited to very large models. The original ToT paper also highlights limitations in complex reasoning and action rollout for smaller models like GPT-3.5. Our results with a simpler greedy search implementation on a 7B parameter LLM (also Llama) align with these expectations. While resource constraints prevented us from exploring larger models (70B Llama on AWS), our qualitative analysis suggests that performance on a 7B model might not be far off. Many of the observed "invalid actions" seem to be minor deviations from task instructions, particularly related to long-term output formatting. We show an example of the agent failing in Fig. 7.

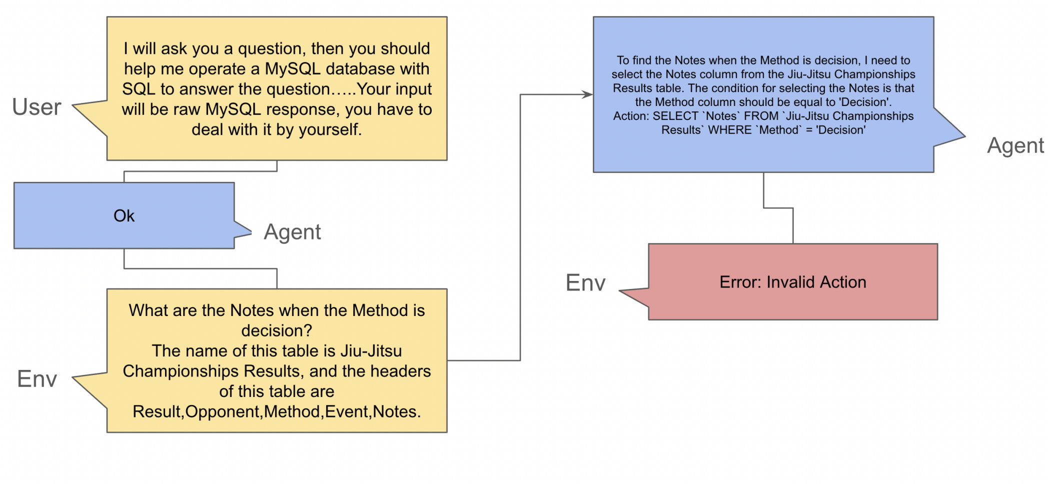

Figure 7: Tree-of-Thought Fails on DBBench

Tree-of-Thought (ToT) poses several challenges for small LLMs, hindering their ability to effectively leverage this framework for complex tasks. Firstly, ToT relies on the LLM's capability to explore and evaluate multiple reasoning paths simultaneously. Small LLMs, with their limited computational resources and memory capacity, struggle to maintain and analyze these branching thought trees. This can lead to them prematurely abandoning promising paths or getting stuck in unproductive branches. Additionally, ToT often requires the LLM to reason about its own reasoning process, a form of metacognition. Smaller models often lack the sophistication to perform this kind of higher-order thinking, making it difficult for them to effectively evaluate and refine their thought trees, ultimately leading to suboptimal solutions or nonsensical actions. This suggests that a certain level of LLM capacity is necessary to handle the cognitive demands of ToT effectively.

13 Future Work

This project explores leveraging large language models as capable agents in diverse digital environments. However, there remain significant avenues for further exploration and advancement. As shown in our results, we were able to improve upon our baselines and our analysis shows that we can achieve even better results using the same techniques on larger models like llama-70B which we could not explore because of compute constraints.

One promising direction is to investigate the application of techniques like Tree of Thought (ToT) and self-reflection with larger language models, which have shown promising results in enhancing reasoning capabilities. Tree of Thought encourages language models to generate step-by-step reasoning chains, breaking down complex problems into intermediate steps. By explicitly reasoning through the problem, the model can arrive at more accurate solutions while providing transparency. Larger language are equipped with emergent abilities and thus can generate more coherent and logically sound reasoning chains, improving performance on complex tasks.

Similarly, self-reflection techniques like Reflexive Agent with Functional Abstractions (RAFA) have demonstrated potential to enhance language models' ability to introspect, reason about their own capabilities, and adapt behavior accordingly. Enabling self-reflection could lead to more robust and trustworthy agents.

Developing multi-turn agent capabilities to engage in interactive dialogues, seeking clarification or gathering additional information to successfully complete tasks is a promising direction. Endowing agents with this ability can enhance robustness and ensure accurate task understanding before execution. Significantly enhancing usability and effectiveness in practical applications.

Investigating techniques for effective task decomposition and hierarchical planning could improve the ability to tackle intricate, multi-step tasks while maintaining coherence and efficiency.

Lastly, developing interpretability techniques to provide insights into language models' decision-making processes becomes increasingly important as they grow in complexity. Understanding how models arrive at outputs, identifying potential biases, and enabling human oversight are crucial for building trustworthy agents.

In summary, future work should focus on Tree of Thought, self-reflection, multi-turn and interactive agent capabilities, task decomposition and planning, and interpretability methods with larger models. Addressing these areas can unlock the full potential of large language models as capable, trustworthy, and transparent agents for diverse digital environments.

14 Conclusion

The primary objective of this project was to explore the capabilities of large language models as intelligent agents capable of interacting with and performing tasks within diverse digital environments, such as operating systems, databases, and virtual worlds. Specifically, we aimed to reproduce the performance of LLM agents in these environments and investigate techniques to augment their decision-making abilities through approaches like fine-tuning, vector representations, and adaptive feedback.

Through a comprehensive literature review, we identified several promising avenues for enhancing LLM agent performance, including novel training strategies, iterative refinement techniques, and instruction-based conditioning methods. Our baseline experiments with state-of-the-art models like Llama 2 7B and 70B demonstrated their potential as capable agents but also highlighted the need for further improvements to match or exceed human-level performance consistently.

To address these challenges, we implemented and evaluated extensions such as instruction tuning and Tree of Thought (ToT) methods. Our results indicate that techniques like instruction tuning, which involves fine-tuning LLMs on a diverse set of task instructions, can significantly improve their performance on specific agent tasks like database querying and operating system interactions. Additionally, the ToT approach, which encourages the generation of step-by-step reasoning chains, showed promise in enhancing the transparency and coherence of LLM decision-making processes.

These findings have significant implications for the development of more capable and trustworthy AI agents. By leveraging the remarkable natural language understanding and generation capabilities of LLMs, combined with targeted fine-tuning and reasoning techniques, we can create agents that can navigate complex digital environments, comprehend and execute intricate instructions, and provide human-interpretable explanations for their actions.

However, our research also highlights several areas for future exploration. Techniques like self-reflection, multi-turn interactive capabilities, task decomposition, and hierarchical planning hold promise for further improving the robustness, adaptability, and reliability of LLM agents. Additionally, developing interpretability methods to provide insights into the decision-making processes of these large-scale models will be crucial for building trustworthy and transparent AI systems.

15 Member Contributions

All team members contributed equally to the project. The exact demarcations are difficult – work was done over zoom pair programming sessions, team huddles and passing files to each other. Due to severe time constraints, any available team member took up the current task at that moment.

References

[1] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. Agenttuning: Enabling generalized agent abilities for llms, 2023.

[2] Wanpeng Zhang and Zongqing Lu. Adarefiner: Refining decisions of language models with adaptive feedback, 2023.

[3] Nicholas Crispino, Kyle Montgomery, Fankun Zeng, Dawn Song, and Chenguang Wang. Agent instructs large language models to be general zero-shot reasoners, 2023.

[4] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36, 2024.

[5] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv: 2308.03688*, 2023.

[6] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.