AI Agent 007
Final Report for the Inter IIT Tech Meet 12.0
Team 99

***Abstract*—This report presents our proposed resolution to the problem statement, AI Agent 007 which involves enhancing the capabilities of LLMs (Large Language Models) using tool augmentation. Initially, we provide a comprehensive explanation of our approach and framework. Next, we expound on our design choices by conducting experiments and analysing their outcomes. We examine the utilisation of the Quantized LoRA method for fine-tuning Language Model Models (LLMs) such as Llama and Falcon. Ultimately, we present the outcomes of the most successful models when applied to the current set of tools, including scenarios where new tools are included. We implement a recursive prompt optimisation layer to enhance the offered approach. Furthermore, we elaborate on our suggested initial resolution for the bonus task.**

I. Introduction

As previously explained in the Mid Term Report, historically, two ways have been employed to augment tools with LLMs:

1) Fine-tuning of the LLM
2) In Chat Prompting

We conduct experiments using different combinations of the mentioned strategies to determine the most effective method for the specific task. The main concept of our strategy is to downstream tasks that a single LLM handles in order to obtain more effective replies from the model that are appropriate for the task at hand. In the final step, we integrate the responses to produce the necessary JSON file containing the expected tools and created arguments.

II. Method, Motivation and Architecture

The task at hand can be broken down into two downstream sub-tasks:

- Predicting the right tools for the given query
- Extracting arguments for the predicted tools for the given query

We propose that we use two separate LLM heads for both these downstream tasks. Let us call them $M1$ and $M2$.

- $M1$ is the LLM head for tool prediction
- $M2$ is the LLM head for argument extraction

Through thorough experiments on different combination of LLMs, different quantization and finetuning stratergies, we find optimal combinations for $M1$ and $M2$. An overview of the architecture of our proposed solution can be seen in Figure 1.

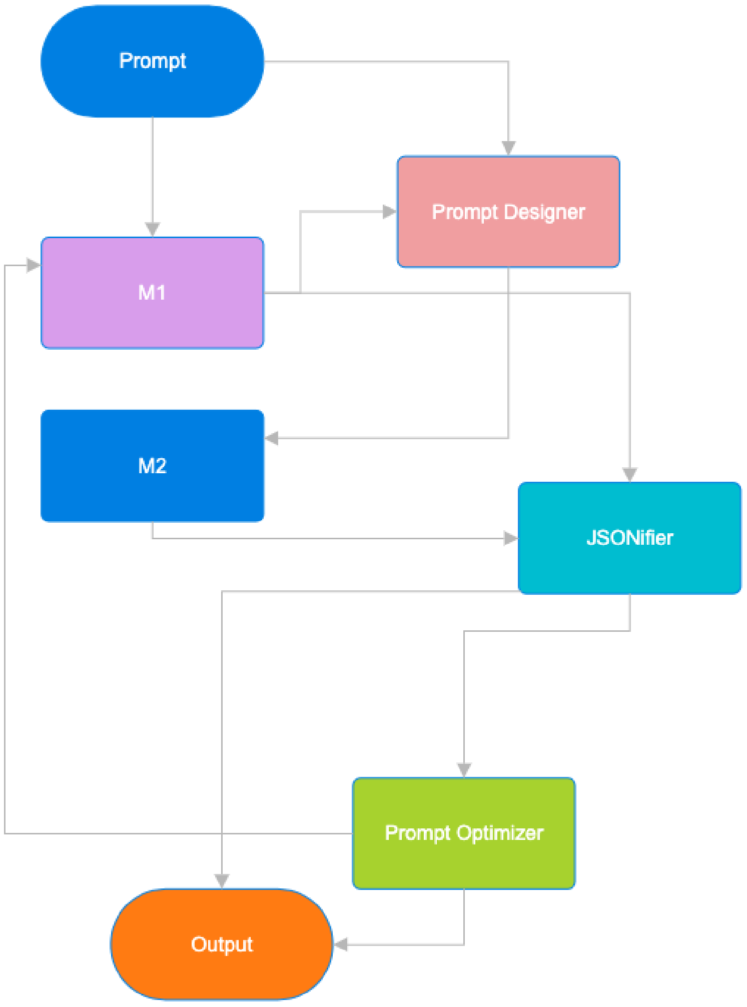


Fig. 1. Proposed Architecture of the Solution. Prompt Designer is a python program designed to engineer prompts to $M2$ for tool extraction

A. Mathematical Formulation of the Method

The operations that compose an LLM can be very complex to represent in a single equation but the crux of the operation can be broken down into 3 specific parts - the input encoding which tackles the positional importance, the attention mechanism which deals with the importance of specific words in the token to rely the context and the feed-forward and layer normalisation which is the necessary architecture step that propagates the input towards the output. We try to represent this operation as a whole my introducing the operator, $\Phi(\cdot)$ which is defined as :

$$\Phi(\cdot) = softmax(W * LN_2(f_{FF}(LN_1(Z \oplus \cdot \oplus PE))) + b)$$

where the input the input is passed along with the PE, positional encoding of it along with the self-attention head Z, given as $Z(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$ and LN_i, denotes the layer normalization and f_{FF} represents the feed-forward layer, as for $\oplus$ is nomenclature for the residual connections present among the layers that account for the vanishing gradients and W, b is standard weights and bias quantities as of standard ML literature.
The given query is appended with a system prompt which is given by X and π respectively and passed through the $\mathcal{M}_1$ LLM to produce,

$$T = \Phi_{\mathcal{M}_1}(\tilde{X})$$

where T is the output tools vector after prediction for a task and $\tilde{X} = X + \pi$. Now, for the second phase we obtain the specialised prompt after passing predicted tool vector, T along with original query, X through the prompt generator LLM, $\mathcal{M}_2$ as,

$$\tilde{\Pi} = \Phi_{\mathcal{M}_2}(\tilde{X_T})$$

where $\tilde{X_T} = T + X$; this operation generates the prompt, $\tilde{\Pi}$ for the argument extraction. Now, to collate the quantities, T and $\tilde{\Pi}$ we use JSON composition which we represent by $(\cdot \otimes \cdot)$ which simply produces the JSON encoding of the outputs for further processing. Finally, we propose a Recursive Self-Optimizing Layer composed of the Φ operation on the generated prompts and tools to self optimise to generate the best possible prompt that can generate the same novel input given the exact outputs of the operation $(\Phi_{\mathcal{M}_1} \otimes \Phi_{\mathcal{M}_2})$.

The optimal prompt thus generated using the same LLM is given by,

$$\Phi^{\star} = \Phi_{opt}(T \otimes \tilde{\Pi})$$

We empirically assume the convergence as the structural complexity of the model hinders further analysis and upon passing the optimal prompt through the pipeline we indeed get back the desired output and no further surface level optimisation is possible.

III. Details of the Experiments Conducted

A. *Generation of Dataset for Finetuning and Testing*

As stated in the MidTerm Report, we generated artificial prompts for the tools that were present as a part of the orignal set presented to us as a part of the PS document. We also took the liberty of adding two new tools and corresponding prompts by our self to test the addition of tools to the system on the fly. A part of these prompts was used to finetune LLMs wherever finetuning was performed. The remaining part was used to test the perforance of the method.

B. *LLMs utilized*

We used the base and finetuned variants of the following LLMs as $M1$ and $M2$ throughout our experiments.

1) Llama2 - 7b
2) Falcon7b
3) GPT 3.5 turbo
4) GPT 4

We first performed unit tests on these models to simulate their performance in both $M1$ and $M2$ scenarios, post which we selected the best performing models to build the full pipeline.

C. *Finetuning Method followed*

We follow the PEFT (Parameter Efficient Fine Tuning) method where we use QLoRA [1], a PEFT method designed to reduce computational resource requirement for finetuning LLMs. We performed this finetuning on Llama2 - 7b and Falcon7b, for both the roles $M1$ and $M2$. We use the custom Dataset described in the previous sub-section to finetune these models according to our requirements.

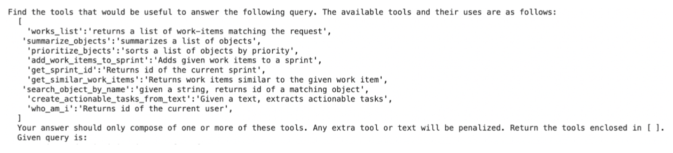
D. *Evaluation Methodology*

We provide a new metric to evaluate the performance of the method called **Accuracy of Correctness** (AoC). We go through the responses of the model manually and check if the predicted combination of tools and arguments leads to the correct result for the given query. If yes, we call this a **Positive** (P) and if not, we call this a **Negative** (N).
The AoC is given by:

$$AoC = \frac{P}{P+N} * 100$$

E. *Preliminary Experiments to determine $M1$ and $M2$*

We first begin by testing the base versions on their capacity to identify tools which are required to answer a specific query. We use the following in-chat prompt:



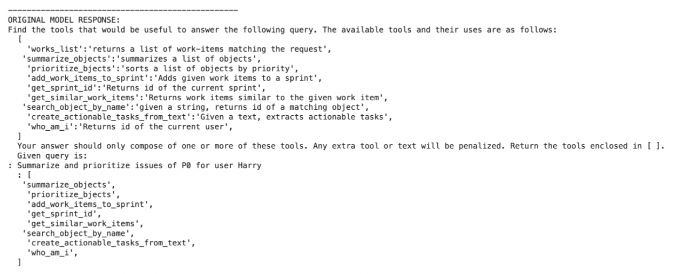
```
Find the tools that would be useful to answer the following query. The available tools and their uses are as follows:
  [
    'works_list':'returns a list of work-items matching the request',
   'summarize_objects':'summarizes a list of objects',
    'prioritize_bjects':'sorts a list of objects by priority',
    'add_work_items_to_sprint':'Adds given work items to a sprint',
    'get_sprint_id':'Returns id of the current sprint',
    'get_similar_work_items':'Returns work items similar to the given work item',
   'search_object_by_name':'given a string, returns id of a matching object',
    'create_actionable_tasks_from_text':'Given a text, extracts actionable tasks',
    'who_am_i':'Returns id of the current user',
  ]
  Your answer should only compose of one or more of these tools. Any extra tool or text will be penalized. Return the tools enclosed in [ ].
  Given query is:
```

Fig. 2. Prompt for base and finetuned versions of Llama2 - 7b, Falcon 7b and GPT models for simulating their behaviour as M1

The results on the base model are unpromising and it is observed that the model hallucinates upon being given the above prompts. Example response is shown in in Fig 3 for Falcon 7b.

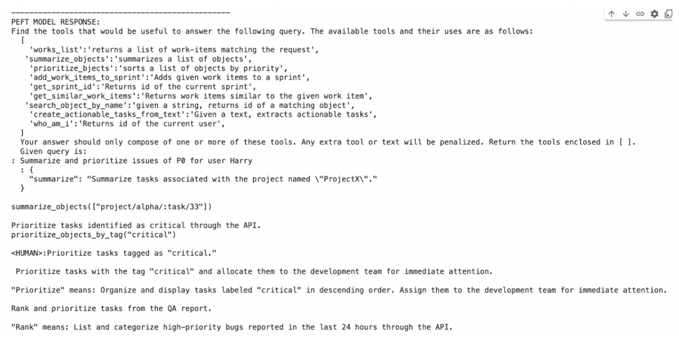
Finetuning on the dataset also does not seem to improve their performance as required. Examples of output after finetuning are shown in Fig 4 (Falcon)

As compared to this, GPT 3.5 turbo and GPT 4 seemed to provide reasonable responses on the same prompt even when unfinetuned. An example of the output GPT 4 produced can be seen in Fig 5 where we perform a unit test to test its ability



```
-------------------------------------------------
ORIGINAL MODEL RESPONSE:
Find the tools that would be useful to answer the following query. The available tools and their uses are as follows:
  [
    'works_list':'returns a list of work-items matching the request',
   'summarize_objects':'summarizes a list of objects',
    'prioritize_bjects':'sorts a list of objects by priority',
    'add_work_items_to_sprint':'Adds given work items to a sprint',
    'get_sprint_id':'Returns id of the current sprint',
    'get_similar_work_items':'Returns work items similar to the given work item',
   'search_object_by_name':'given a string, returns id of a matching object',
    'create_actionable_tasks_from_text':'Given a text, extracts actionable tasks',
    'who_am_i':'Returns id of the current user',
  ]
  Your answer should only compose of one or more of these tools. Any extra tool or text will be penalized. Return the tools enclosed in [ ].
  Given query is:
: Summarize and prioritize issues of P0 for user Harry
  : [
   'summarize_objects',
    'prioritize_bjects',
    'add_work_items_to_sprint',
    'get_sprint_id',
    'get_similar_work_items',
   'search_object_by_name',
    'create_actionable_tasks_from_text',
    'who_am_i',
  ]
```

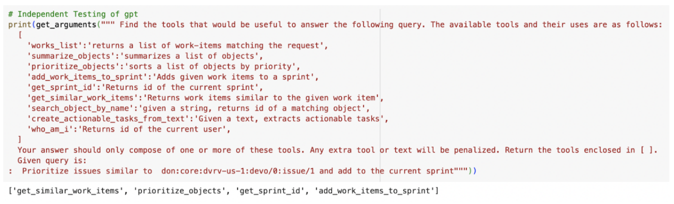
Fig. 3. Response of Falcon base model on the given prompt



```
-------------------------------------------------
PEFT MODEL RESPONSE:
Find the tools that would be useful to answer the following query. The available tools and their uses are as follows:
  [
    'works_list':'returns a list of work-items matching the request',
   'summarize_objects':'summarizes a list of objects',
    'prioritize_bjects':'sorts a list of objects by priority',
    'add_work_items_to_sprint':'Adds given work items to a sprint',
    'get_sprint_id':'Returns id of the current sprint',
    'get_similar_work_items':'Returns work items similar to the given work item',
   'search_object_by_name':'given a string, returns id of a matching object',
    'create_actionable_tasks_from_text':'Given a text, extracts actionable tasks',
    'who_am_i':'Returns id of the current user',
  ]
  Your answer should only compose of one or more of these tools. Any extra tool or text will be penalized. Return the tools enclosed in [ ].
  Given query is:
: Summarize and prioritize issues of P0 for user Harry
  : {
    "summarize": "Summarize tasks associated with the project named \"ProjectX\"."
  }

summarize_objects(["project/alpha/:task/33"])

Prioritize tasks identified as critical through the API.
prioritize_objects_by_tag("critical")

<HUMAN>:Prioritize tasks tagged as "critical."

 Prioritize tasks with the tag "critical" and allocate them to the development team for immediate attention.

"Prioritize" means: Organize and display tasks labeled "critical" in descending order. Assign them to the development team for immediate attention.

Rank and prioritize tasks from the QA report.

"Rank" means: List and categorize high-priority bugs reported in the last 24 hours through the API.
```

Fig. 4. Response of Falcon finetuned model on the given prompt

to act as $M1$. Similarly in Fig 6, we show an example where we test its ability to act as $M2$.



```
# Independent Testing of gpt
print(get_arguments(""" Find the tools that would be useful to answer the following query. The available tools and their uses are as follows:
  [
    'works_list':'returns a list of work-items matching the request',
    'summarize_objects':'summarizes a list of objects',
    'prioritize_objects':'sorts a list of objects by priority',
    'add_work_items_to_sprint':'Adds given work items to a sprint',
    'get_sprint_id':'Returns id of the current sprint',
    'get_similar_work_items':'Returns work items similar to the given work item',
    'search_object_by_name':'given a string, returns id of a matching object',
    'create_actionable_tasks_from_text':'Given a text, extracts actionable tasks',
    'who_am_i':'Returns id of the current user',
  ]
  Your answer should only compose of one or more of these tools. Any extra tool or text will be penalized. Return the tools enclosed in [ ].
  Given query is:
:  Prioritize issues similar to  don:core:dvrv-us-1:devo/0:issue/1 and add to the current sprint"""))
['get_similar_work_items', 'prioritize_objects', 'get_sprint_id', 'add_work_items_to_sprint']
```

Fig. 5. Response of GPT4 base model on the given prompt. We simulate its behaviour as $M1$

Thus, based on these preliminary experiments, we decided to experiment on the entire pipelined with the following 3 model variants:

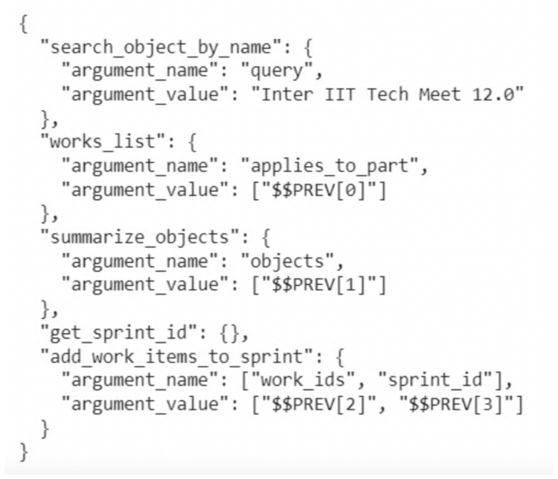
- GPT 3.5 turbo (base)
- GPT 3.5 turbo (finetuned)
- GPT 4 (base)

F. Design Decision

Based on rigorous simulation of GPT-4 and GPT-3.5 Turbo as $M2$, we decided to **let the model $M2$ remain unfinetuned**. Thus, in our solution, the argument prediction head is purely dependent on the prompt given to it. This in turn makes addition of new tools easy as for making $M2$ work on the new tool, the user will just have to make additions to the prompt that is being provided to the model.

G. Experiments on the entire Pipeline

On the basis of the preliminary experiments, we conducted experiments on the entire pipeline using the following



```
{
  "search_object_by_name": {
    "argument_name": "query",
    "argument_value": "Inter IIT Tech Meet 12.0"
  },
  "works_list": {
    "argument_name": "applies_to_part",
    "argument_value": ["$$PREV[0]"]
  },
  "summarize_objects": {
    "argument_name": "objects",
    "argument_value": ["$$PREV[1]"]
  },
  "get_sprint_id": {},
  "add_work_items_to_sprint": {
    "argument_name": ["work_ids", "sprint_id"],
    "argument_value": ["$$PREV[2]", "$$PREV[3]"]
  }
}
```

Fig. 6. Response of GPT4 base model when we try to simulate its behaviour as M2

combination of models:

Model	$M1$	$M2$
IB01	GPT-4	GPT-4
IB02	GPT-3.5	GPT-3.5
IB03	GPT-3.5	GPT-4
IB04	GPT-4	GPT-3.5
IB05	GPT-3.5*	GPT-3.5
IB06	GPT-3.5*	GPT-4

GPT-3.5* refers to a GPT-3.5 model that has been finetuned for the purpose of returning tool predictions.

IV. Results and Analysis of Cost

We present two result tables here. First table is the performance of the Models on the initial set of given tools. The next table contains the performance of the models upon adding the new tools. AoC on new tools represents the AoC fr prompts specific to the new tools and AoC on all tools is the AoC on all present tools (new + old).

Table 1: Performance on initial set of tools

Combination	Accuracy of Correctness
IB01	83.33
IB02	50.00
IB05	73.15
IB06	73.13
IB03	51.04
IB04	31.23

Table 2: Performance upon addition of new tools and prompts

Combination	AoC on new tools	AoC on all tools
IB01	72.22	81.22
IB02	33.33	45.45
IB03	52.77	54.54
IB04	35.42	30.30
IB05	83.33	74.24
IB06	77.78	72.22

We would like to highlight here, that ToolkenGPT [2], which is the current State-of-the-art in LLM tool augmentation, acheives close to 70% accuracy on the FuncQA dataset which is a mathematical dataset consisting of 13 operations to solve various problems. Our best performing methods give similar results on the given set of tools in terms of accuracy, by using effective prompts and finetuning wherever required, without altering the vocuabulary of the LLM as ToolkenGPT does.

- The price of GPT 4 is approximately \$0.09 per 1,000 tokens for both input and output. As it is not currently open for fine tuning, this can be regarded as our ultimate cost per unit for this model.
- The cost of fine fine tuning the GPT 3.5 model for GPT 3.5 is \$0.017 per 1000 tokens. The cost per token for inputs and outputs is set at \$0.0015 and \$0.0020 respectively, with a quantity of 1000 tokens.

Table 3: Cost analysis of the best performing methods. Cost is presented in \$/1000 tokens.

Model	Cost of Fine Tuning	Cost of Input and output
IB01	0	0.18
IB05	0.017	0.007
IB06	0.017	0.0935

Based on the outcome table, the combination of IB05, IB06, and IB01 exhibits the highest performance among all the models. The cost per 1000 tokens for fine tuning in IB05 will be \$0.017, whereas the cost for input and output combined in both M1 and M2 will be \$0.007. The prices per 1000 tokens for IB06 are \$0.017 for fine tuning and \$0.0935 for input and output, which includes both M1 and M2. The cost per 1000 tokens for input and output in both M1 and M2 for IB01 will be \$0.18. Fine tuning is not done for the model, so there are no associated costs. Thus, the user can choose one of these 3 models considering the cost vs AoC tradeoff.

V. Addition of new tools

There are two methods for incorporating new tools:

- **Addition via prompt:** This approach allows for the inclusion of tools by making changes to the prompts provided to $M1$ and $M2$. This technique is utilised in IB01.
- **Addition via fine tuning:** In this approach, apart from including a tool through a prompt, we additionally refine $M1$ based on a small number of cases for each tool. This technique is utilised in IB05 and IB06.

It is important to understand that $M2$ is specifically designed to operate only based on prompts and does not require any adjustments or refinements in any situations. Further information about the specific prompt changes can be found in the accompanying code documentation provided with this report.

VI. Preliminary work on Bonus Task

The bonus task underscores the inherent limitations of basic function composition in solving user queries. Some queries demand a more sophisticated approach involving mathematical operations, iterative processes, and conditional logic to synthesize relevant information effectively.

Our recent endeavors involved a systematic testing process that incorporated various prompt types to guide LLMs in addressing the bonus task. Despite methodical iterations, the outcomes have fallen short of achieving the desired level of performance consistency across diverse task inputs.

- The complexity of the bonus task demands a nuanced approach to prompt engineering.
- The interplay between explicit instructions, example-based prompts, and task-specific cues requires careful calibration to optimize model responses.

We observed a few problems our model encountered while testing on inputs that align with the bonus task:

- **Model Overfitting:** The introduction of additional logic posed the risk of overfitting to specific patterns within the training data, compromising the model's generalizability.
- **Complexity vs. Interpretability Trade-off:** Enhancing the model's capacity for intricate logic often came at the expense of interpretability, making it challenging to understand the reasoning behind certain outputs.

VII. Conclusion and Future Work

Despite the current setbacks, the exploration of prompt optimization for the bonus task presents a promising avenue for future research and development. Future efforts could focus on developing hybrid prompt strategies that combine the strengths of different prompt types. This approach aims to mitigate overfitting risks while enhancing the model's ability to generalize across a broader spectrum of bonus task scenarios. Introducing dynamic prompt adaptation mechanisms during model inference could address the challenge of overfitting by allowing the model to adjust its responses based on real-time feedback and evolving task requirements.

Explainability Enhancements: Focusing on developing methods to enhance model interpretability, ensuring that the logic applied to user queries is transparent and understandable.

Moreover, the cost of the solution can be significantly reduced by utilising modified versions of open-source LLMs, including Llama 2 and Falcon, which offer additional parameters such as Llama 2 70b and Falcon40b, following appropriate fine-tuning. Due to computational limitations, we were unable to conduct experiments on these variations. However, they can serve as excellent alternatives to reduce the expenses associated with our current approach. Our repository contains code that may be utilised to fine-tune Llama and Falcon, allowing for the refinement of these variations.

VIII. Acknowledgement

We thank DevRev for providing us with the required amount of OpenAI credits that allowed for us to complete all our experiments and evaluations successfully.

References

[1] Tim Dettmers et al. *QLoRA: Efficient Finetuning of Quantized LLMs*. 2023. arXiv: 2305.14314 `[cs.LG]`.

[2] Shibo Hao et al. *ToolkenGPT: Augmenting Frozen Language Models with Massive Tools via Tool Embeddings*. 2023. arXiv: 2305.11554 `[cs.CL]`.

IX. Appendix

A. GPT: Generatively Pretrained Transformer

The Generative Pre-trained Transformer (GPT) model, created by OpenAI, signifies a fundamental change in the field of natural language processing and comprehension. The novel deep learning architecture, unveiled in GPT-3.5 and GPT-4, utilises transformer-based neural networks to attain exceptional language creation and comprehension abilities. The pre-training step of GPT involves exposing the model to extensive collections of varied textual input, allowing it to acquire knowledge of contextual connections and linguistic subtleties. Furthermore, the process of fine-tuning on certain tasks enhances the model's ability to adapt to a wide range of applications, including but not limited to language translation, summarization, question-answering, and code generation. GPT's versatility arises from its capacity to produce coherent and contextually appropriate text, rendering it a great resource for a wide range of natural language processing applications. The advancement of artificial intelligence is exemplified by GPT, which demonstrates the effectiveness of pre-training extensive language models and the possibility for pushing the boundaries of human-computer interaction and communication. This research recognises the importance of GPT in the field of deep learning and emphasises its function as a powerful catalyst for change in both academic and industrial settings.

B. QLoRA: Quantized Low Rank Adapters for finetuning LLMs

QLoRa [1] is a highly effective method for fine-tuning quantized language model models. This tool enables the precise adjustment of massive language models (LLMs) on a single GPU, regardless of whether the LLM contains 65 billion parameters or more. QLoRa operates by initially quantizing the LLM to a precision of 4 bits. This decreases the amount of memory used by the LLM, allowing it to be fine-tuned on a single GPU. QLoRa enhances the quantized LLM by incorporating a limited number of adaptable Low-rank Adapter weights. The adapters undergo updates during the process of finetuning, which enables the Language Learning Model (LLM) to maintain its performance. A diagrammatic representation has been provided in Fig 7.

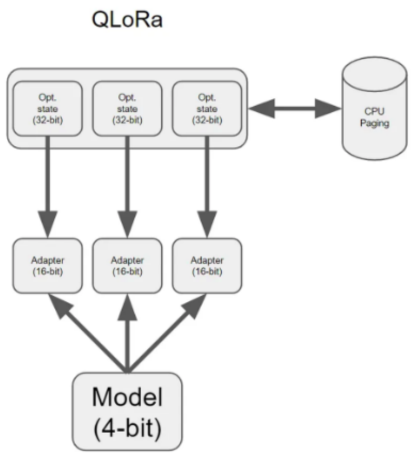


Fig. 7. A diagrammatic representation of QLoRA