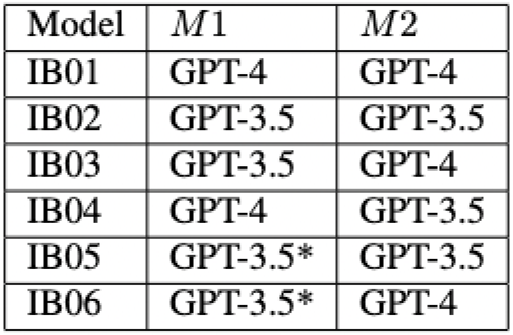
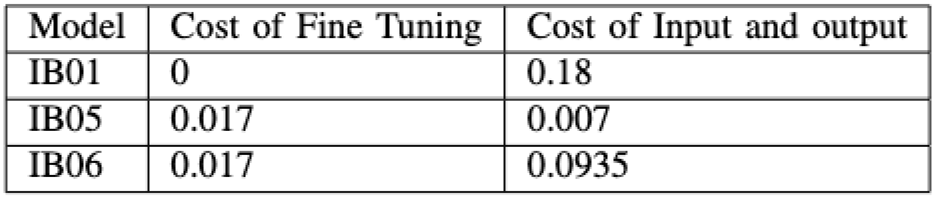
Team 99 : Documentation

Introduction:
The code is organized in 3 folders and accompanied with this documentation. The folder “final_solutions” contains the 3 best performing models. The “inference_notebooks” folder contains the inference notebooks for all other combinations of models as given in the table below.



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

The three best performing models with their cost analysis are as follows. These are present in the “final_solutions” folder.



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

The CSVs for base_model_testing and novel_model_testing are added in the csvs folder.
base_model_testing: prompts on the given tools., novel_model_testing: prompts on the given tools +novel tools added to the repertoire.
Note: Examplar addition of new tools has been added to the notebook IB01_novel.ipynb.
Naming convention:
IB01.ipynb → Notebook on given tools
IB01_novel.ipynb → Notebook on given + additional tools
Similar for all notebooks.
Instructions to run notebooks:

Requirements to run the notebooks: The user must have a valid OpenAI API key with enough credits to access GPT4 and GPT3.5 and finetune them.

- All notebooks from IB01 to IB06 can be run as Jupyter Notebooks. Each notebook runs in a free instance of Google Colab on GPU runtime.

- **Addition of query texts:** A list queries[] has been declared in every notebook. The user is required to add the query strings to this list before running the models on them.
- Appropriate comments are present for clarification. Similar instructions can be followed for other notebooks.(IB01-06).

Addition of New Tools:

- In IB01, addition of tools can happen simply by altering prompts. This has been demonstrated in the notebook IB01_novel.ipynb in the functions: get_tools() and analyse_query().
- For notebooks IB05,06, we require to finetune the first GPT 3.5 instance. This finetuning is described in the GPT_finetuning.ipynb in the “finetuning_attempts” folder.

For example, we have the following two tools:

```
'get_previous_sprint':'Returns the sprint id of the previous sprint',
   'return_top_k_items':'Returns the top k items from the given list of items',
```

We will make the following changes in the code:
In the function: get_tools(): (Additions are highlighted on Green)

```
def get_tools(query):
 system_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_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',

   'get_previous_sprint':'Returns the sprint id of the previous sprint',
   'return_top_k_items':'Returns the top k items from the given list of items',

 ]
 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 """

 user_prompt = f""": {query} : """

 final_prompt = system_prompt + "\n" + user_prompt
 messages=[{
     "role":"user",
     "content":final_prompt
 }]

 responses=openai.ChatCompletion.create(
     model=model,
     messages=messages,
     temperature=0
 )

 return responses.choices[0].message['content']
```

In the function analyse_query(): (Additions are highlighted in green)

```
def analyze_query(tools_list, query_text):
   tools_purpose = {
       'works_list': 'Returns a list of objects matching the request',
       'summarize_objects': 'Summarizes a list of objects',
       'prioritize_objects': 'Returns a list of objects sorted by priority',
       'add_work_items_to_sprint':'Adds the given objects to the sprint',
       'get_sprint_id':'Return the id of the current sprint',
       'get_similar_work_items':'Returns a list of objects that are similar to the given object',
       'search_object_by_name':'Given a search string, returns the id of a matching object in the system of record',
       'create_actionable_tasks_from_text':'Given a text, extracts actionable text The text from which the actionable string insights, and creates tasks for them, which are kind of a work item',
       'who_am_i':'Returns string_id of current user',
       'get_previous_sprint':'Returns the sprint id of the previous
sprint',
       'return_top_k_items':'Returns the top k items from the given list
of items',
   }

   tools_arguments = {
       'works_list': ['applies_to_part: Array of strings to filter works
relevant to', 'created_by: Takes array of strings and filters work created
by users in the array', 'issue.priority: Array of strings to filter issues
with given priorites in the array', 'issue.rev_orgs: Array of strings to
filter issues for the organizations provided in the array', 'limit:
integer providing the maximum number of works to return', 'owned_by: Array
of strings to filter issues owned by users specified in the array',
'stage.name: Array of strings to filter work in the stages provided in the
array', 'ticket.needs_response: Boolean value telling if a ticket needs a
response','ticket.rev_org: Array of strings to return tickets associated
with the given strings', 'ticket.severity: Array of strings to filter
issues with given severity in the array', 'ticket.source_channel: Array of
strings to filter for ticklets of the provided channels in the array',
'type: Array of strings with allowed values: [issue, ticket, task] Filters
for work of the provided types' ],
       'summarize_objects': ['objects: List of object ids to summarize'],
       'prioritize_objects': ['objects: List of objects to prioritize'],
       'add_work_items_to_sprint': ['work_ids: List of objects to be
added', 'sprint_id: Id of the sprint'],
       'get_sprint_id': [],
       'get_similar_work_items': ['work_id: id of work item to find
similar items to'],
       'search_object_by_name': ['query: String to search for'],
       'create_actionable_tasks_from_text': ['text: Text to create
actionable tasks from'],
       'who_am_i': [],
       'get_previous_sprint':[],
       'return_top_k_items':['objects: List of objects sorted by
priority', 'k: Number of items to be returned']
   }
   relevant_purposes = {tool: tools_purpose[tool] for tool in tools_list
if tool in tools_purpose}
   relevant_arguments = {tool: tools_arguments[tool] for tool in
tools_list if tool in tools_arguments}

   output_string = f"The given query utilizes the following tools:
{tools_list}. "
   output_string += f"The arguments of the tools and their description  is
as follows. Format is 'argument_name:Purpose of argument':
{relevant_arguments}. "
   output_string += f"The purpose of the tools is as follows:
{relevant_purposes}. "
   output_string +="Note that the words issues, objects and work_items
have been used interchangably"
   output_string += f"Find the values arguments for the given tools from
the following text:\\ {query_text} \\"
   output_string += "Just return the value of the arguments, do not return
anything else. In case you need to use the output of the previous tool as
an input to the next tool, you can name it as $$PREV[i], where i is the
index of the tool starting from 0. Return answer in nested JSON format
with separate JSONS in one JSON for each tool named after the tool itself.
The keys are: argument_name and argument_value. Every argument need not
have a value. But every tool taking an argument must take atleast one
argument. Only find values for relevant arguments."

   return output_string
```

Similar changes can be made to IB05 and IB06 for addition of new tools along with the finetuning of the first model.

Fine Tuning attempts on Llama and Falcon:

- ipynb files have been added to show our efforts on finetuning Llama and Falcon using QLoRa with 4 and 8 bit quantization. These can also be used to finetune Llama and Falcon models with higher parameter counts.