National College of Ireland

Configuration Manual

MSc Research Project
MSc in Cyber Security MSCCYBETOP

Aivaras Prudnikovas
Student ID: 23277742

School of Computing
National College of Ireland

Supervisor: Raza Ul Mustafa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Aivaras Prudnikovas
Student ID:	23277742
Programme:	MSCCYBETOP **Year:** 2025
Module:	MSc (Research) Practicum Part 2
Supervisor:	Raza Ul Mustafa
Submission Due Date:	Monday 11th August 2025 by 2pm
Project Title:	LLM based prompt injection detection accuracy and its relationship to the context size
Word Count:	1158 **Page Count:** 6

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Aivaras Prudnikovas

Date: 2025-08-02

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Aivaras Prudnikovas
Student ID: 23277742

1 About

To evaluate the proposed hypotheses the test environment was set up. To be able to execute on objectives it was necessary to run tests and evaluations against the running Large Language Models (LLMs).

There are a couple of core components that need to be configured to be able to run the tests. First, is a test execution environment which contains test definitions and inputs which satisfy the objectives. Then, there are remote LLM services and a local one for running small models that are capable to fit into RAM.

2 Prerequisites

Test environment requires a computer with administrative rights to be able to install the required software. The steps in this manual require a Unix based system but it should be possible to adapt them to the Windows or various flavours of Linux. The hardware should have at least 16 GB of RAM and a powerful CPU to be able to run some tests on a locally running LLM.

3 Test environment

Test files and the input data are hosted in the publicly accessible GitHub repository[1]. The files should be downloaded first, they are version controlled, use the default main branch when cloning with *git*:

```
$ git clone https://github.com/ivarprudnikov/llm-prompt-injection-detection-
accuracy tests
$ cd tests
```

Source files contain a detailed *README.md* which has the latest information about the project directories and how to run the tests. **Please consult it before running the tests**.

It will be necessary to have a *Python 3.12* installation[2] on the computer and the *venv*[3] module to be able to isolate dependencies required for the tests. Create a new local isolated environment and install dependencies defined in the *requirements.txt* file:

[1] https://github.com/ivarprudnikov/llm-prompt-injection-detection-accuracy
[2] https://www.python.org/downloads/
[3] https://docs.python.org/3.12/library/venv.html

```
$ python3 -m venv venv
$ source venv/bin/activate
(venv) $ pip install --upgrade pip
(venv) $ pip install -r requirements.txt
```

To be able to run tests using remote LLM providers the dedicated SDKs expect a set of configured environmental variables. There is a sample configuration that needs to be copied and filled in with the correct values for the tests to successfully execute:

```
$ cp .env .env.local
$ cat .env.local
OPENAI_API_KEY=
MISTRAL_API_KEY=
AZURE_AI_URL=
AZURE_AI_KEY=
AZURE_AI_API_VERSION=
GEMINI_API_KEY=
```

Each of the variables must be set which will be used in the test execution, the details on how to get these values described below in the chapter about LLM providers.

3.1 Clear existing test outputs

The outputs that were captured in the evaluation were added to the version control history and will be present after cloning the repository. To avoid them from interfering with the new test execution make sure to either delete them or backup to another folder:

```
# To backup
$ mkdir backup
$ mv data/output_* backup
$ mv data/overview.html backup

# Or, to delete
$ rm -rf data/output_* data/overview.html
```

3.2 Running tests

Once the values are set in .env.local the test execution can begin. The Jupyter[4] notebook files (tests) use a naming convention "*run_${modelName}_${contextSize}.ipynp*". The tests would need to be executed for each model with an increasing context size, e.g. for model *gpt4* in order *run_gpt4_0k.ipynb, run_gpt4_1k.ipynb, run_gpt4_5k.ipynb, run_gpt4_10k.ipynb, run_gpt4_50k.ipynb* (start with *0k* and end with *50k*). It is important to start with *0k* as it creates an output file used in other tests.

To run these test files, it is necessary to have the software which is capable to open and execute them. Originally the tests were carried out using VS Code editor[5] with added Jupyter extension[6]. It is possible to use other Jupyter notebook editors to run these tests if necessary.

[4] https://jupyter.org/
[5] https://code.visualstudio.com/Download
[6] https://marketplace.visualstudio.com/items?itemName=ms-toolsai.jupyter

Open the first test to run (e.g. *run_gpt4_0k.ipynb*) and select the *kernel* and point it to the python environment created earlier (*venv/bin/python*). Then, to run the code select *Run All* in Jupyter notebook (Fig 1). Observe for any issues that might be present when executing code blocks, if there are any, mitigate the described error and run again.

The test output will be written to the JSON file in the data directory using naming convention "output_${size}_${modelName}.json", e.g. *output_basic_gpt_4_1.json*. Possible sizes are: *basic* for *0k*, then *increased_1k*, *increased_5k*, *increased_10k*, *increased_50k*.

Note: Tests of the same model and/or LLM provider would not run successfully if executed in parallel due to rate limits.

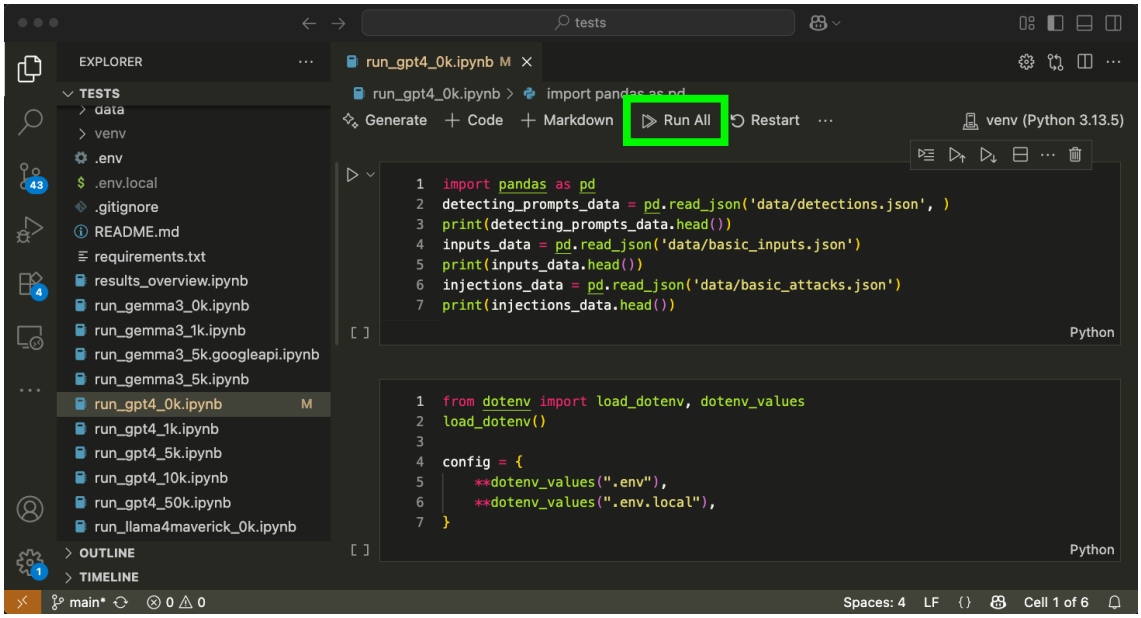

Fig 1. VS Code user interface showing one of the test notebook files loaded and the button Run All highlighted.

4 Remote LLM service providers

An account with each provider is necessary to obtain the API key and other information in case of Azure AI. It will be necessary to setup billing to be able to process a large volume of tokens required by the tests.

4.1 Open AI

Sign up on Open AI platform[7] then create your project and a new secret key (Fig 2). Secret key will be in the form of *sk-abc* and will need to be set in *.env.local* file, e.g. *OPENAI_API_KEY=sk- ****.

[7] https://platform.openai.com

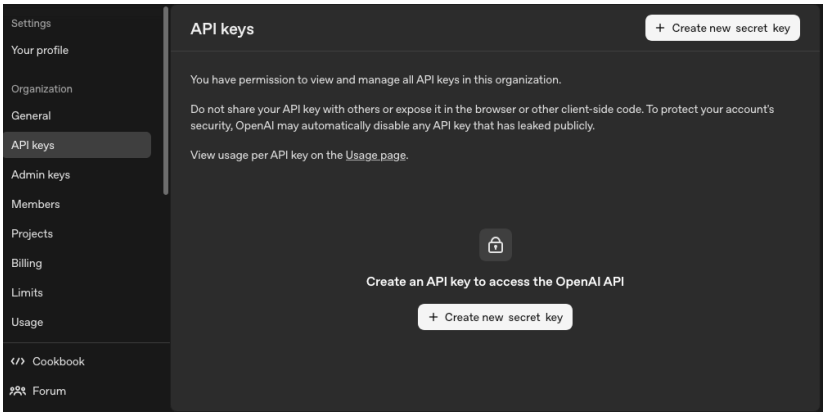

Fig 2. Open AI API keys management page has an option to create a new key.

In billing, add at least €100 in credits and top it up if when tests exhaust them and/or begin receiving repeating rate limiting errors. It will be necessary to wait for at least a week after first payment for the usage tier to be increased in the account, the tests will require *Tier 2*[8] at a minimum.

Note: tests (5 notebooks) at the time of the research used €135.90 credits in total.

4.2 Mistral

Sign up to Mistral AI[9] and create an API key (Fig 3). This key will need to be saved in *.env.local* file, e.g. *MISTRAL_API_KEY=*****.

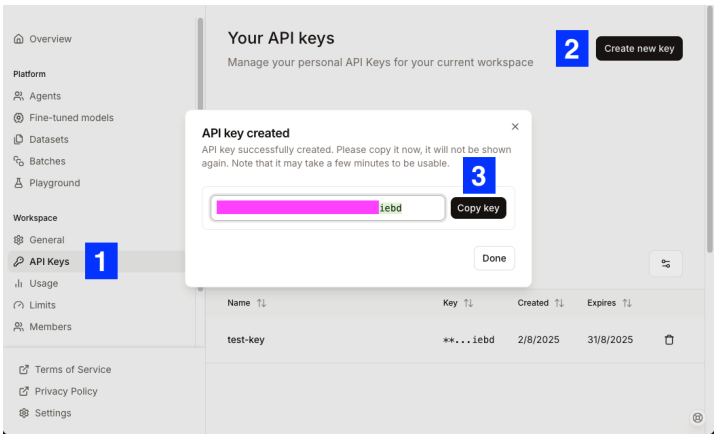

Fig 3. Numbered steps in Mistral show how to create a new API key.

Set up billing in settings and add a card to be charged.

Note: Mistral tests (9 notebooks) at the time of the research costed €113.67 in total.

[8] https://platform.openai.com/docs/guides/rate-limits#usage-tiers
[9] https://console.mistral.ai/

4.3 Azure AI

LlaMa Maverick model was chosed to be run in Azure AI but it is necessary to deploy that model first.

Create an Azure account[10] and setup a new subscription. Then make sure to create a Azure AI Foundry resource[11]. Once created follow the link to the the Azure AI Foundry portal (Fig 4).

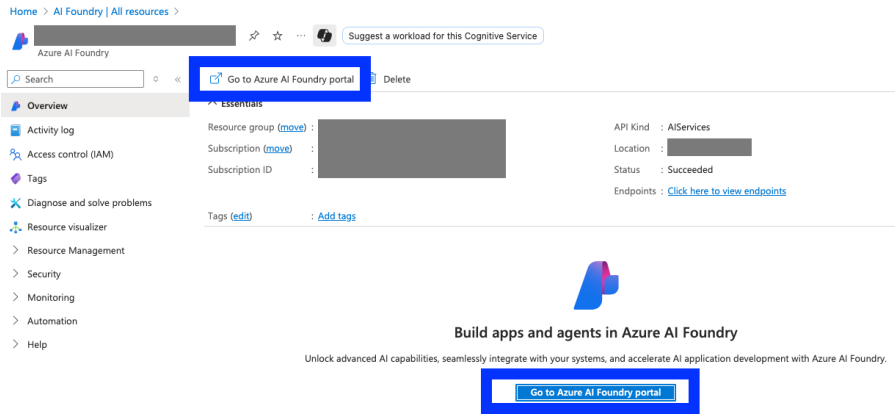

Fig 4. Links to Azure AI Foundry visible in the resource view.

Within Azure AI Foundry portal open "Models + Endpoints" from the side menu, then click on "Deploy model" and find "Llama-4-Maverick-17B-128E-Instruct-FP8" (Fig 5). Once found configm the deployment. If the exact same model does not exist then deploy another similar model.

If the deployed model differs from the one which is configured in the notebook it will be necessary to update it in each notebook. There are three notebooks that test Llama model and each reference it by name in one of the code cells, they will all need to be updated, e.g.:

```
response = client.complete(
        model="Llama-4-Maverick-17B-128E-Instruct-FP8",
        messages=[SystemMessage(content=sys_prompt),
UserMessage(content=input_text)],
        temperature=0.1,
        max_tokens=16
)
```

After the model deployment three variables will need to be set in .env.local file: key, url and API version. These values will be available from the deployed model view, e.g.:

```
AZURE_AI_URL=https://*****.services.ai.azure.com/models
AZURE_AI_KEY=******
AZURE_AI_API_VERSION=2024-05-01-preview
```

[10] https://azure.microsoft.com
[11] https://learn.microsoft.com/en-us/azure/ai-foundry/

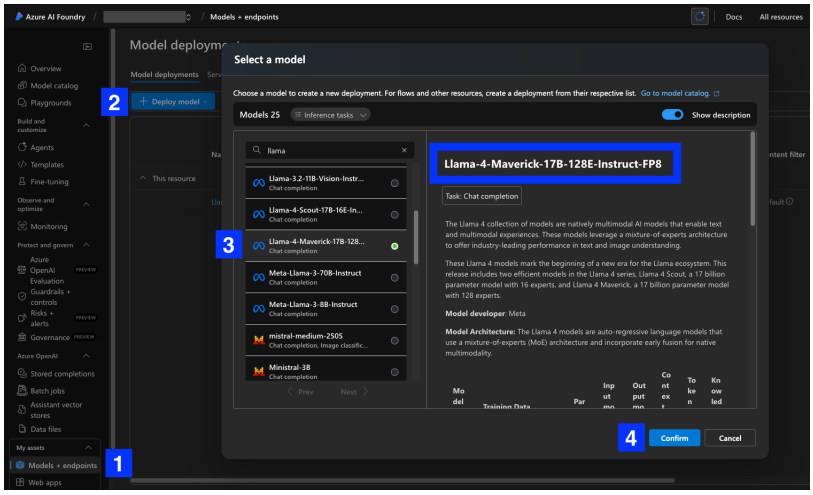

Fig 5. Steps marked to deploy the Llama model in Azure AI Foundry.

4.4 Gemini

Open Google AI Studio[12] and click on Get API key to procure a key. This key will need to be added to the *.env.local* configuration file under *GEMINI_API_KEY*. If no free credits are available, it will be also necessary to setup billing, but it does not matter which tier is used because Gemma 3 12B model which is used in tests is throttled in all tiers according to Google documentation[13].

In the case if the API constantly returns HTTP status 429, increase the sleep seconds between tests in the notebook that runs them, e.g. set it to five seconds *time.sleep(5)*.

Note: The same model is being used when testing on the local LLM server.

5 Local LLM server

One of the tests that uses small model *Gemma3:12b* requires a locally running LLM. Install Ollama application[14] which will allow to download and runn the model locally.

```
$ ollama pull gemma3:12b
$ ollama run gemma3:12b
```

Once the model is running I is possible to execute tests in the *run_gemma3_0k.ipynb* notebook.

6 Processing results

Finally, after running all the tests that produce output files in the data directory, the last step is to process those results to produce the charts and tables used in research paper. Run *results_overview.ipynb* notebook and inspect the outputs after each code cell.

[12] https://aistudio.google.com
[13] https://ai.google.dev/gemini-api/docs/rate-limits
[14] https://ollama.com/