Leveraging Generative AI For Test Automation With The Power Of Natural Language Processing (NLP)

Ahsan Muzammil, Stephen Wynn-Williams, Jacob Brodersen, Vera Pantelic, and Mark Lawford.

Department of Computing and Software, McMaster University, Hamilton, Canada. Centre for Software Certification (McSCert), McMaster University, Hamilton, Canada

Introduction – What is NLP?

- NLP is a branch of AI dedicated to the interaction between human language and computers, including tasks like translating natural language (e.g. English sentences) into formal structures (e.g. code).
- This summer, I worked on a collaborative project between McSCert and an industrial partner.
- We were provided with a sample of industrial test cases (formatted in XML) and utilized generative AI to transform the natural language descriptions from these test cases into a formal, code-like structure which we compared to the original implementation.

Natural Language	Formal Language
Check foobar = $X and wait (1000 ms)	READ(foo_bar, X) WAIT(1000)

Project Toolkit

Version Control and Collaboration:

- **GitLab**: For managing our code repository and facilitating collaboration among team members.

Development Environment:

- **VS Code**: As my primary code editor for writing and debugging code.

Programming Language:

- **Python**: The main programming language used for our project.

Natural Language Processing Frameworks:

- **Hugging Face**: Utilized for accessing pre-trained language models.

Experiment Management Tracking:

- **MLflow**: A platform used to manage the machine learning lifecycle, including experimentation, reproducibility, and deployment.

Scripting:

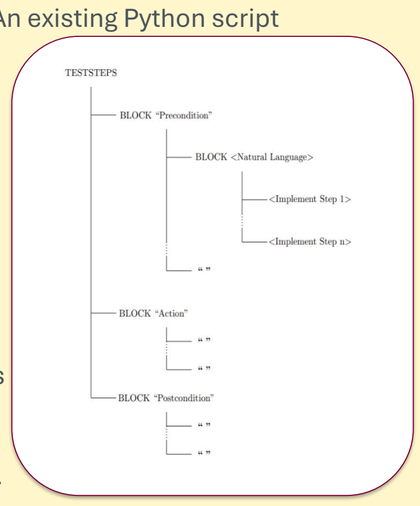
- **Bash Scripts:** Used to automate and run the experiments.

Large Language Models (LLMs) Used:

- **Gemma/Gemma-IT**
- **GPT-J**
- **Mistral/Mistral-IT**

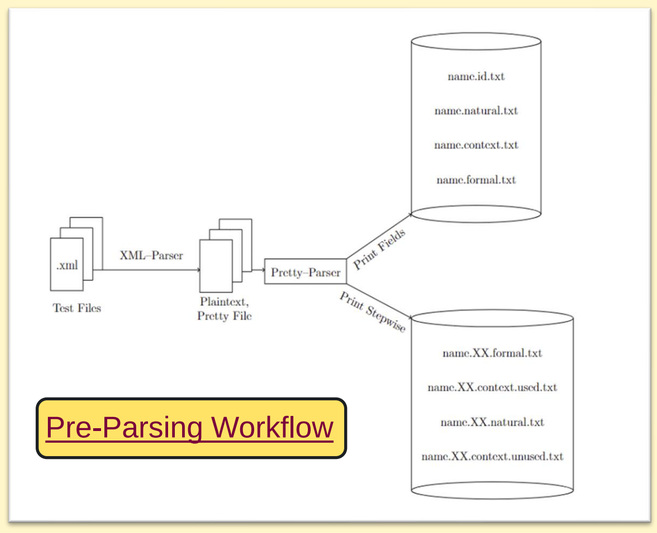
Pre-Parsing (XML Parser)

- **XML Data Extraction:** An existing Python script was used to parse the provided industrial test cases, extract relevant information, and consolidate it into a single plaintext file named "pretty." This approach was used for easy manual modification of the dataset.



XML Structure

- **Pretty File Parsing:** Refined a Python script to separate contents of "pretty" files into distinct files based on sections like context, natural language, formal language, ID or organized the data into stepwise files with separate used and unused context, formal and natural languages for each step.



Pre-Parsing Workflow

Generate Formal Language

- Takes pre-parsed data, prefix, and produces a prompt. Feeds inputs to a specified pre-trained model.
- The model processes the input and produces the corresponding formal language output.
- Compute the CHRF score by comparing the expected and actual strings. Then log the results to MLflow and run a post-processing script to incorporate additional metrics if necessary.

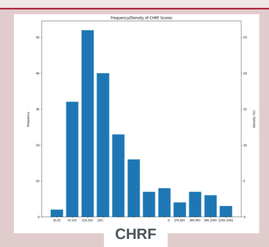
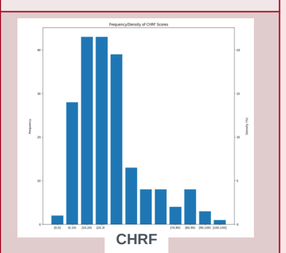
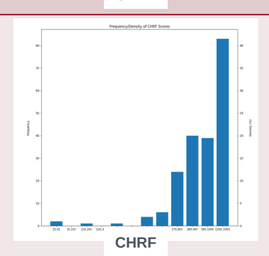
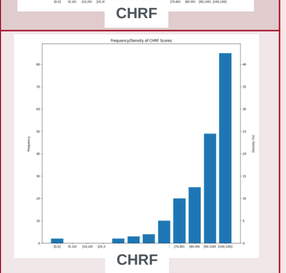
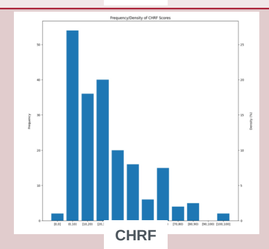
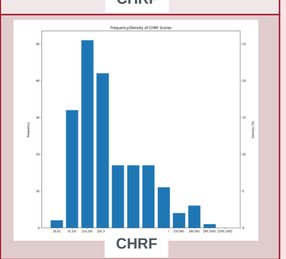
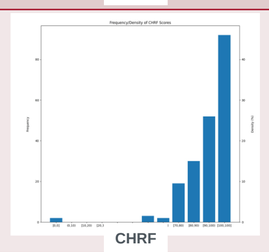
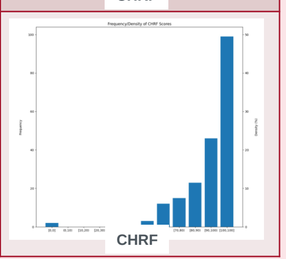
Manipulating Generative AI Inputs:

Sort the context section in random or alphabetical order. For random shuffling, we use a seed to ensure the reproducibility of the shuffle.

Rearrange the prompt by altering the order of natural language and context within the input provided to the generative AI.

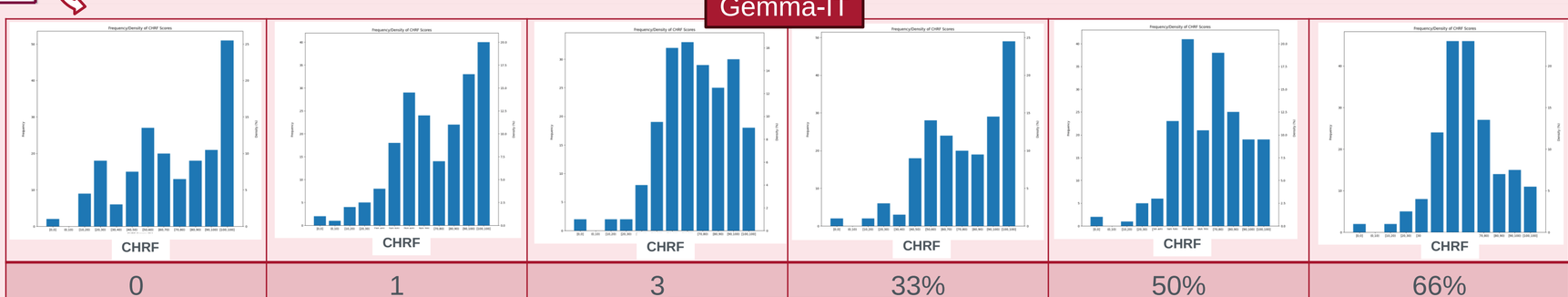
Progressively add irrelevant context to the used context and observe its impact on the generative AI's ability to produce the expected result

Observations From Input Manipulation

Gemma Model	Shuffled	Alphabetical
Context First (Full Context)		
Context First (Relevant Context only)		
Natural Language First (Full Context)		
Natural Language First (Relevant Context only)		

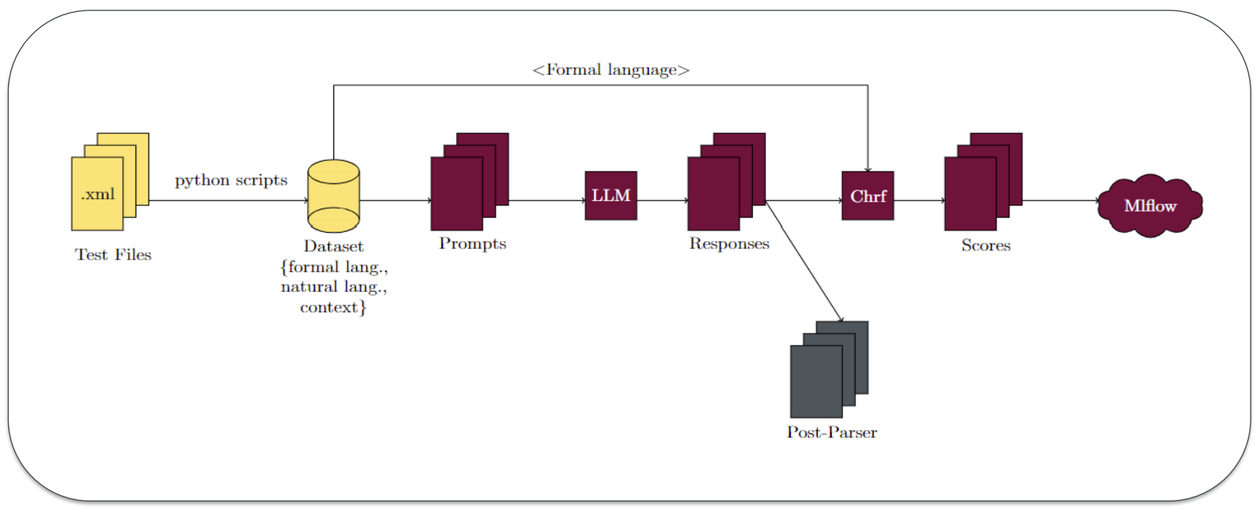
Irrelevant Context	Gemma-IT (CHRF Score)
0 Irrelevant Entries	66.44
1 Irrelevant Entries	66.33
3 Irrelevant Entries	67.02
33% Irrelevant Entries	67.67
50% Irrelevant Entries	63.62
66% Irrelevant Entries	60.63

Gemma-IT



Irrelevant Context

Structure of The Project



Post-Parsing (Formal Language Parser)

- This script processes the formal language output by the model and displays it on the command line as a JSON string.
- **Input:** Formal language (provided either as a string or from a file)
- **Output:** Prints to the command line a list of dictionaries, with each dictionary representing a command, or nested commands.
- **Example:**

```
* LOOP(4)
  * READ(varB, 0)
  * WRITE(varA, val)
```

```
[
  {cmd: "loop", count: "4", inner: [
      { cmd: "read", mapping: "varB", value: "0" },
      { cmd: "write", mapping: "varA", value: "val" },
    ],
  },
]
```

Ahsan Muzammil
Third-Year Software Engineering Student

McMaster University, Department of Computing and Software
McMaster Automotive Resource Centre, MARC 230

Email: muzammia@mcmaster.ca

BRIGHTER WORLD