MASTER PROJECT

eXercise Open Call Project

eXercise Documentation

NOESISHUB & 8 BELLS

June 10, 2025

Table of Contents

eXRercise sub-project of the MASTER first Open Call (OC)

The following documentation addresses the integration of assets developed using Unity Engine and the Unity Editor, for the purposes of the MASTER OC project **eXercise**.

One of the components of the **eXercise Training System** is the creation of VR-based scenarios. The library described below enables dynamic VR scenarios for: - fire incidents (by creating a virtual fire effect at a robotic arm/device) - malfunction scenarios at a specified robotic arm or device

This component accepts structured decisions output from an LLM prompt, used by a trainer. The result is a VR scenario presented to a trainee. It is distributed as a Unity-native DLL library.

Pilot Project Report: NLP-based Trainer

1 Objective

The primary aim of this pilot project was to create a streamlined, interactive interface enabling trainers to input high-level textual descriptions (prompts) regarding training scenarios involving virtual fires and robotic arm malfunctions. These inputs are processed by advanced language models (LLMs) to generate scenario recommendations, identify malfunctions, and support training decisions.

2 Overview of System

The system has been developed using Streamlit and comprises two key interface sections:

1. Robotic Arm Selector Tab

2. AI Advisor Tab

2.1 Robotic Arm Selector Tab

This tab provides users (trainers) with the ability to:

- Select an LLM model, supporting both OpenAI and Hugging Face APIs (e.g., GPT-4o, Zephyr, Mistral).
- Dynamically input the number of robotic arms for the session.
- Submit textual prompts describing training scenarios.
- Enter distance values for each robotic arm relative to the trainee.

The system then:

- Processes the prompt and distances using the selected LLM.
- Returns structured JSON output indicating:
  - Which robotic arm may be experiencing a virtual fire.
  - Which arm is potentially malfunctioning.
  - An explanation for the AI's decision.
- Displays example prompts to guide user input.
- Persists all session data for later analysis.

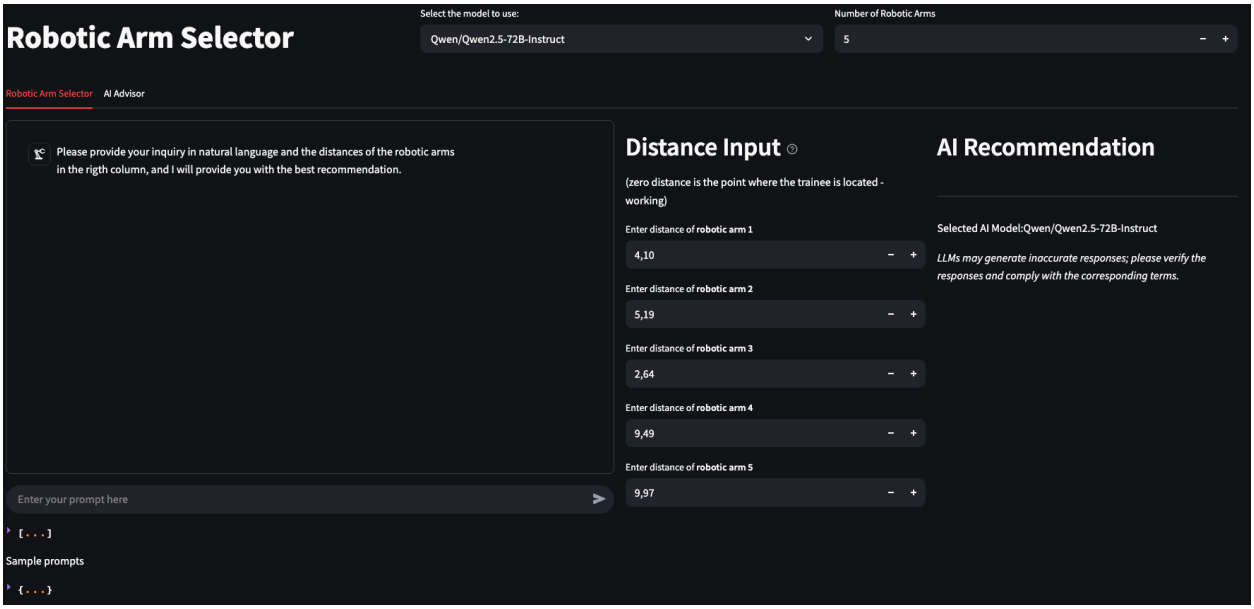


Figure 1: Robotic Arm Selector Tab

2.2 AI Advisor Tab

This tab supports two advanced functions:

1. Session Stress Result Retrieval: Connects to an external stress classification API to retrieve the participant's emotional/stress performance based on physiological data.
2. Performance Report Generation: Uses the LLM to generate a markdown report incorporating:
    - Session metadata (date, time, participant ID)
    - Robotic arm distances
    - Malfunction and fire detection summary
    - Stress analysis results
    - Analytical tables and conclusions

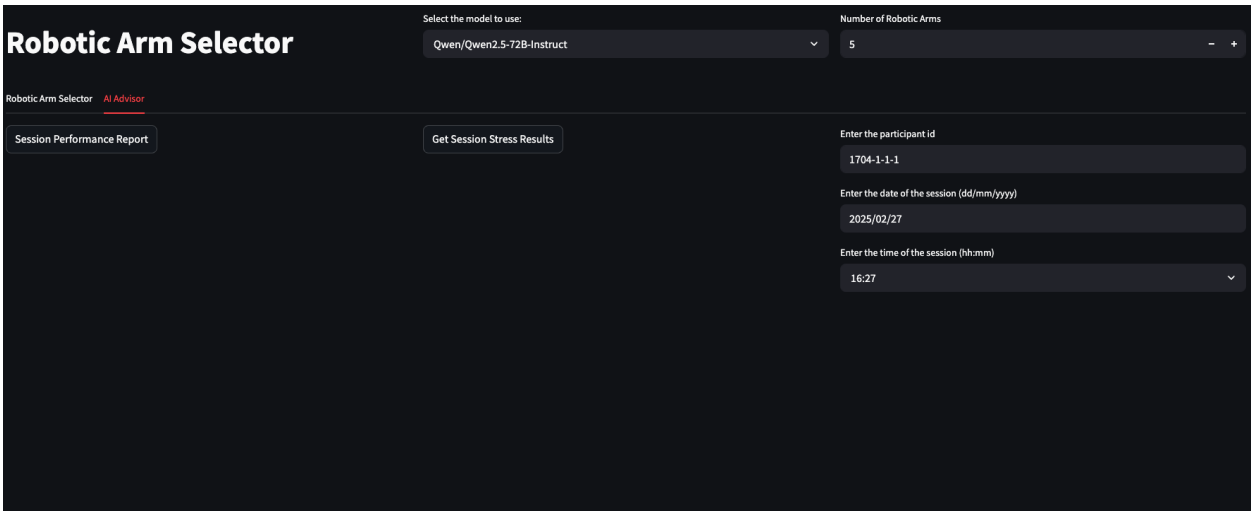


Figure 2: AI Advisor Tab

3 System Components

3.1 Natural Language Interface

The platform leverages large language models (via OpenAI and Hugging Face) for prompt processing. The system dynamically constructs a structured prompt containing:

- Trainer's textual input
- Robotic arm distance values
- Request for identification of robotic arm associated with virtual fire and malfunction

3.2 Distance Input Panel

Users can input precise numerical distances for each robotic arm. These values are used by the LLMs to make scenario inferences.

3.3 Model Integration

Models supported include:

- GPT-4o, GPT-4o-mini (OpenAI API)
- Zephyr-7b-beta, Mistral-7B-Instruct-v0.3, Qwen2.5-72B-Instruct (Hugging Face Inference API)

The list of models can be extended to include others.

3.4 Stress Analysis

The system queries a stress analysis API to retrieve a participant's classification result for the session. It matches timestamps and participant IDs to fetch the correct entry.

4 Sample Output Format


```
{
  "Robotic_Arm_Virtual_Fire": 1,
  "Robotic_Arm_Malfunctioning": 2,
  "Reason_of_selection": "Robotic Arm Number 1 is the closest to the base with a distance of 1.43, whic
}
```


Generated with OpenAI API (gpt-4o).

5 Usability and User Experience

- The interface is intuitive and provides real-time feedback.
- Prompts with fewer than 20 characters are disallowed to prevent vague input.
- Prompt inputs are stored and reused for report generation.
- Users can view AI model outputs in both JSON and Chat UI formats.

6 Challenges and Recommendations

- Challenge: Occasional JSON parsing failures from LLM responses.
  Recommendation: Implement retry logic and more constrained prompt formatting.
- Challenge: Ambiguity in prompt interpretation when language is imprecise.
  Recommendation: Enhance prompt scaffolding and allow iterative refinement.

7 Conclusion

The pilot project successfully demonstrated the feasibility of translating natural language trainer inputs into actionable XR training scenarios. The integration of multiple LLMs, combined with real-time data capture and post-session reporting, creates a powerful tool for enhancing safety training simulations.

Robotic Arm & AI Advisor Installation & Deployment Guide

This guide walks you through deploying the **Robotic Arm & AI Advisor** Streamlit app to a Hugging Face Space and configuring a Hugging Face dataset `XYZ` for storing LLM result JSON files.

---

1. Prerequisites

Make sure you have:

- A Hugging Face account
- `git` installed on your machine
- `Python >= 3.11` installed
- A valid Hugging Face token (`HF_TOKEN`) with write permissions
- (Optional) An OpenAI API key (`OPENAI_API_KEY`) if using OpenAI models

---

2. Create a Hugging Face Space

1. Go to Create a New Space.
2. Fill in:
   - **SDK**: `Streamlit`
   - **Space Name**: e.g., `robotic-arm-selector`
   - **License**: Select one appropriate
   - **Visibility**: `Public` or `Private`
3. Click **Create Space**.

---

3. Create and Add Access Tokens

A. Create Hugging Face Token (`HF_TOKEN`)

1. Visit Hugging Face Tokens.
2. Click **New token**, name it `HF_TOKEN`, set permissions to **Write**.
3. Copy the generated token securely.

B. Add Hugging Face Token to Space

1. In your Space, go to **Settings** > **Secrets**.
2. Add a new secret:
   - **Name**: `HF_TOKEN`
   - **Value**: your copied token

C. (Optional) Add OpenAI Token

If using OpenAI models (`gpt-4o`, etc.), add:

- **Name**: `OPENAI_API_KEY`
- **Value**: your OpenAI API key

---

4. Prepare and Upload Your Files

You need these files in your Space repository:

`RoboticArm.py`

Your main Streamlit app. Ensure it includes code for uploading LLM result JSON to a Hugging Face dataset (see Section 7). See source content

`requirements.txt`

List dependencies:

```
streamlit
huggingface_hub>=0.31.4
sentence-transformers
openai
pydantic
requests
```

`README.md`

Configure your Space:

```yaml
title: RoboticArm
sdk: streamlit
emoji: (see file to render)
colorFrom: pink
colorTo: indigo
sdk_version: 1.44.1
app_file: RoboticArm.py
pinned: false
python_version: '3.11'
```

---

5. Upload and Deploy

Option A: Upload through Web UI

1. In your Space, go to **Files and versions**.
2. Click **Add file > Upload files**.
3. Upload `RoboticArm.py`, `README.md`, and `requirements.txt`.

Option B: Upload via Git

```
# Clone your space
git clone https://huggingface.co/spaces/<your-username>/robotic-arm-selector
cd robotic-arm-selector

# Add your files
cp /path/to/RoboticArm.py .
cp /path/to/README.md .
cp /path/to/requirements.txt .
```

```
# Commit and push
git add .
git commit -m "Add initial app files"
git push
```

Once uploaded, the Space will build and deploy automatically. View and test your app via the Space URL.

---

6. Create and Configure the Hugging Face Dataset `XYZ`

To store LLM result JSON files from the app, create a HF dataset repository. Name of the dataset can be anything! As long as you replace XYZ with your name of the dataset.

A. Create the dataset repository

You can do this locally or via the website.

1. **Via CLI (locally)**:

   ```
   huggingface-cli login
   huggingface-cli repo create XYZ --type dataset
   ```

   Replace `XYZ` with your chosen name. Choose `public` or `private` as needed.

2. **Via Web UI**:

   - Go to New dataset.
   - Name it `XYZ`, set visibility.

3. **Record the full repo ID**: e.g. `your-username/XYZ`. This will be used in code.

B. Add HF token for dataset access

- In your local environment (if testing outside Spaces), set:

  ```
  export HF_TOKEN="<your_hf_token>"
  ```

- In the Space, you already added `HF_TOKEN` as a secret (Section 3).

---

7. Integrate Dataset Upload in `RoboticArm.py`

Modify your Streamlit app to upload LLM results (`response_llm_json.json`) to the `XYZ` dataset.

In order to divert to your dataset, locate the following code snippet and edit it appropriately to match your own Hugging Face dataset.

```
api = HfApi()
with open("response_llm_json.json", "rb") as fobj:
   api.upload_file(
         path_or_fileobj=fobj,
         path_in_repo="response_llm_json.json",
         repo_id="your-username/XYZ",
         repo_type="dataset",
         commit_message="Upload generated file",
         token=os.getenv("HF_TOKEN")
   )
```

Replace `repo_id` with your dataset ID/ dataset name.

---

8. Testing the Upload Flow

1. Deploy or run the app in your Space.
2. In the UI, enter a valid prompt to trigger LLM call and JSON creation.
3. Observe the Streamlit UI for success message.
4. Visit your Hugging Face dataset page: https://huggingface.co/datasets/your-username/XYZ to see uploaded JSON files.

---

9. Appendix: Example Workflow Summary

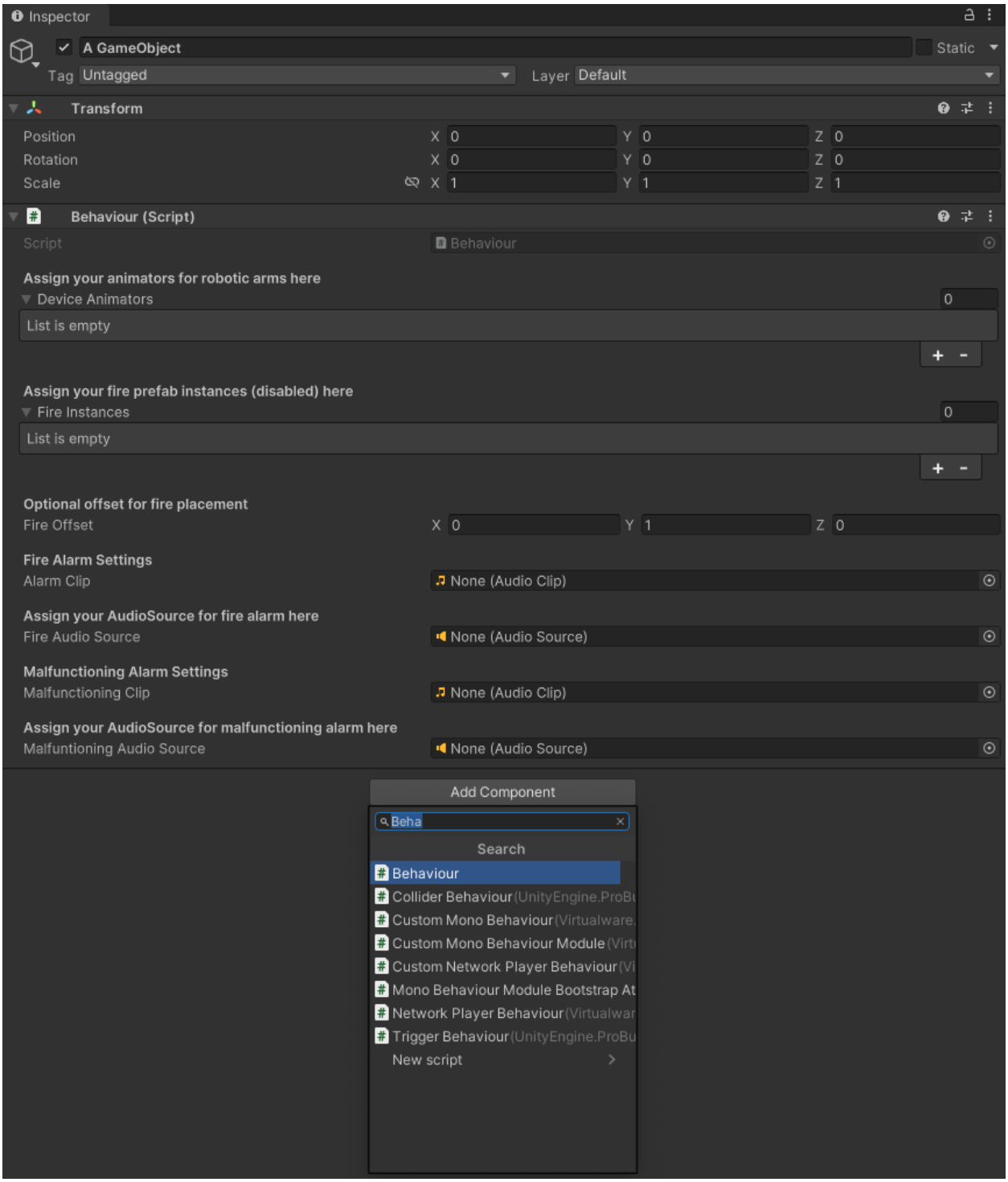
1. **Create Space** and add secrets (`HF_TOKEN`, optionally `OPENAI_API_KEY`).
2. **Create HF dataset** named `XYZ`, note `your-username/XYZ`.
3. **Prepare files** (`RoboticArm.py`, `requirements.txt`, `README.md`) with upload logic.
4. **Upload files** to Space via UI or Git, then deploy.
5. **Test** the app: trigger LLM, observe upload success in UI.
6. **Verify** dataset contents on HF Hub.

XR Scenario Creator

1 Components Implemented

Two key components have been developed: - **Behaviour** component - **HuggingFaceJsonFetcher** component

The `Behaviour` is designed to accommodate the virtual robotic arms used in your Unity scene. You can assign it via the Unity Inspector, as shown below:



Example showing the Behaviour component fields in the Unity Inspector.

---

2 Behaviour Component Fields Overview

All fields listed below are customizable for flexibility across different environments and training goals.

Field Name	Type	Description
`deviceAnimators`	Animator[]	Robotic Devices with Animators attached
`fireInstances`	GameObject[]	Prefabs for fire effects
`fireOffset`	Vector3	Offset to place fire effects
`alarmClip`	AudioClip	Sound to play during fire incident
`fireAudioSource`	AudioSource	Audio source for fire alarms
`malfunctioningClip`	AudioClip	Sound to play during malfunction incident
`malfuntioningAudioSource`	AudioSource	Audio source for malfunction alerts

2.1 Behaviour Component Field Descriptions

2.1.1 `deviceAnimators` (Animator[]) A list of Animator components. Assign any GameObject (e.g., robotic arms) that contains an Animator. If no Animator is present, it will not work.

2.1.2 `fireInstances` (GameObject[]) A pool of disabled fire prefabs placed in the scene. The script will activate one randomly during a fire event.

2.1.3 `fireOffset` (Vector3) Offset where the fire prefab appears relative to the device. Default is `(0, 1, 0)`.

2.1.4 `alarmClip` (AudioClip) The sound that plays during a fire incident. Use a looping alarm or suitable alert.

2.1.5 `fireAudioSource` (AudioSource) Audio source used to play `alarmClip`.

2.1.6 `malfunctioningClip` (AudioClip) Audio clip triggered when a malfunction event occurs.

2.1.7 `malfuntioningAudioSource` (AudioSource) Audio source used to play the `malfunctioningClip`.

---

3 Unity Integration Guide – HuggingFace JSON Fetcher

This section explains how to use the `HuggingFaceJsonFetcher` Unity component alongside `Behaviour` to create automated, LLM-driven VR scenarios using structured JSON input.

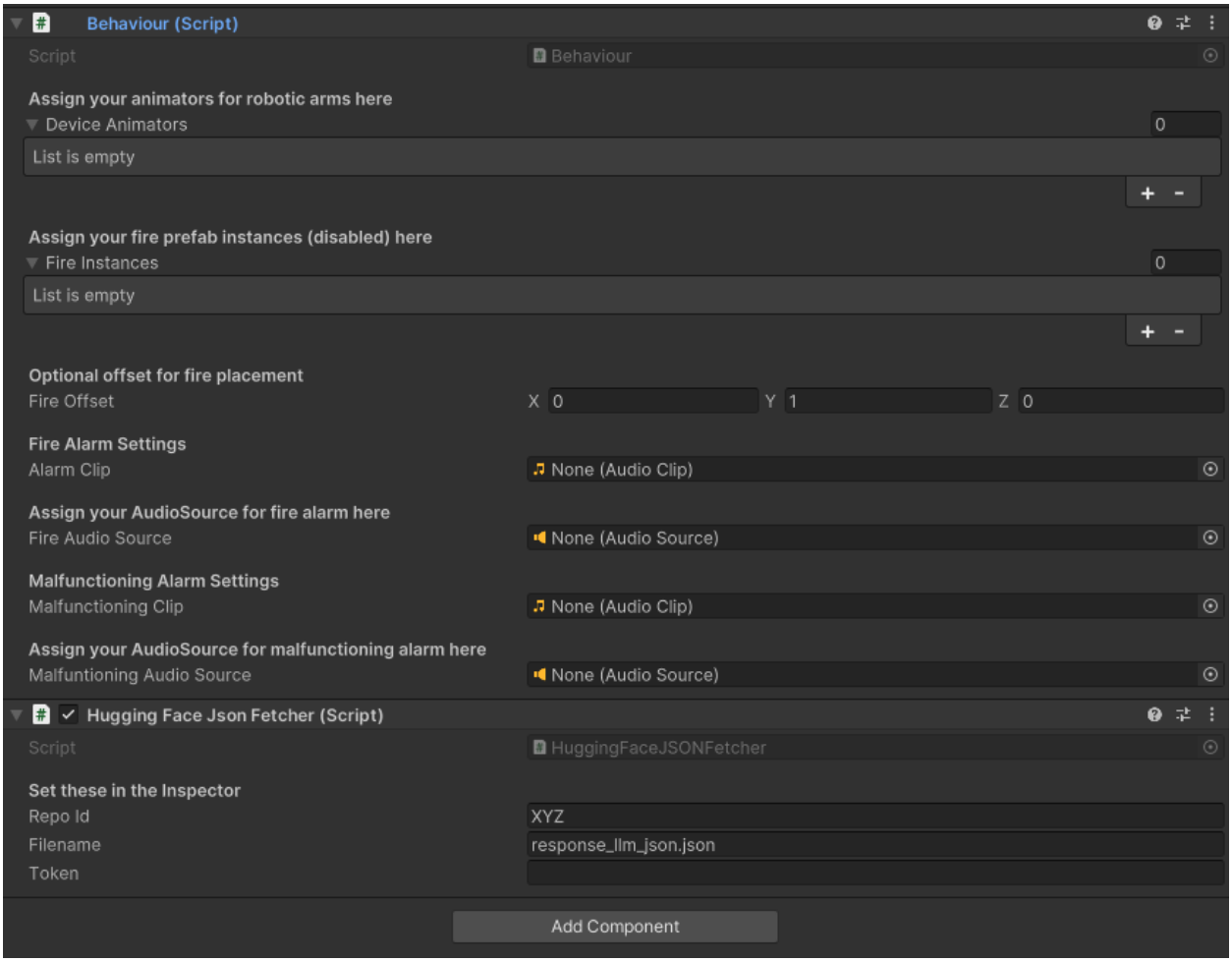
The trainer promets in the provided LLM, the desired scenario (fire, malfuncationing incident or even both) and the LLM return a selection of the selected robotic arm which will virtually be set on fire. When the traineed start the VR application, the library retrieves the structured ouput of the LLM, which indicated the robotic devices to be set on fire, malfunction or in case of a combination (a selection of max. 2 robotic devices) of the incidents which coresponding robotic device suffers which disaster.

An example structured output is shown here, presenting also the JSON structure of the ouput:

```
{
  "Robotic_Arm_Virtual_Fire": 1,
  "Robotic_Arm_Malfunctioning": 2,
  "Reason_of_selection": "Robotic Arm Number 1 is the closest to the base with a distance of 1.43, whic
}
```

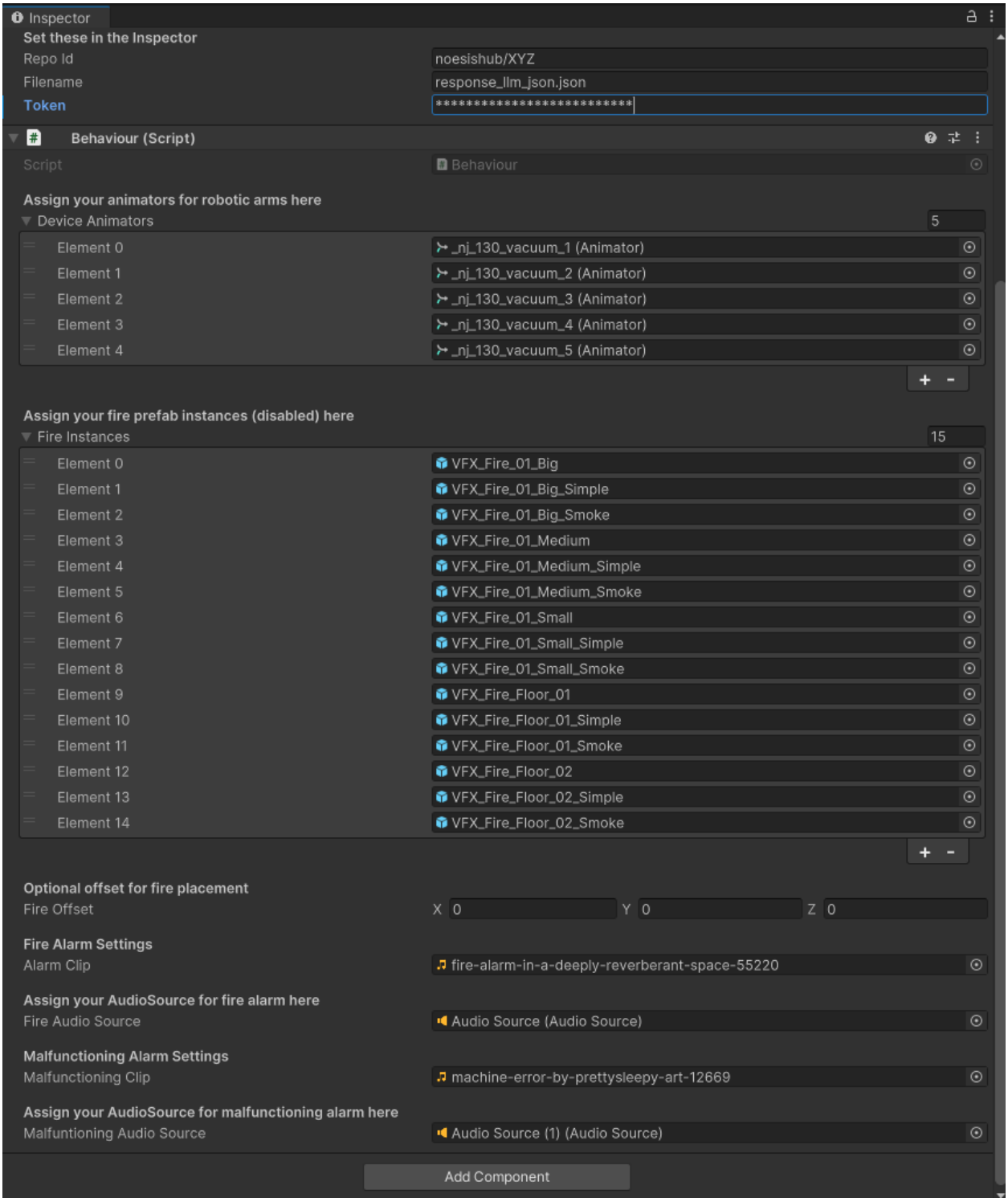
This component, retrieves the LLM structured output from a dataset repository, which must be created in HuggingFace (see the corresponding documentation). In order to successfuly retrieve this information, we have to correctly set the following fields, show in the table below:

Parameter	Type	Description
repoId	string	Hugging Face dataset repository ID (e.g., username/dataset_name).
filename	string	JSON file name inside the dataset.
token	string	(Optional) Hugging Face access token. Required for private datasets.



Example showing the HuggingFaceJsonFetcher component fields in the Unity Inspector.

Below we show how a fully configured Behaviour and HuggingFaceJsonFetcher component looks like.



Example showing how both components can be configured in the Unity Inspector. This is a fully functional configuration.

3.1 HuggingFaceJsonFetcher Field Descriptions

3.1.1 `repoId` (string) The repoId is the name of the created dataset in Hugging Face (this is also the name that must be used in the LLM module in HugginFace). No links or any url of any kind is needed, only

the profile name followed with the dataset name e.g. my-user-name/dataset-name. Inside this repository there must be at least one file (default can be the file with name response_llm_json.json) in which the LLM will write its structured output.

3.1.2 `filename` (string) The name of the file the LLM writes the structured output in the designated dataset (see above field).

3.1.3 `token` (string) In order to access the Hugging Face dataset, we need to create a token from the Hugging Face user account in order for the library to be authenticated and authorized to access the file mentioned above (same logic as in github, gitlab, etc). Refer to this link for additional information on this topic. Be aware to set the correct priviledges, otherwise the library will now be able to access the file, resulting in a 401 error response.

> **BE CAREFULL not to publish or accidentally share the generated token. Line other tokens, is must not be publically available!**