AIGENT_JS

Contributers

Phan Trung Duc - 21125035: Backend developer
Nguyen Dinh Ngoc Tri - 21125065: Frontent developer
Le Truong Tho - 21125158: Frontend developer

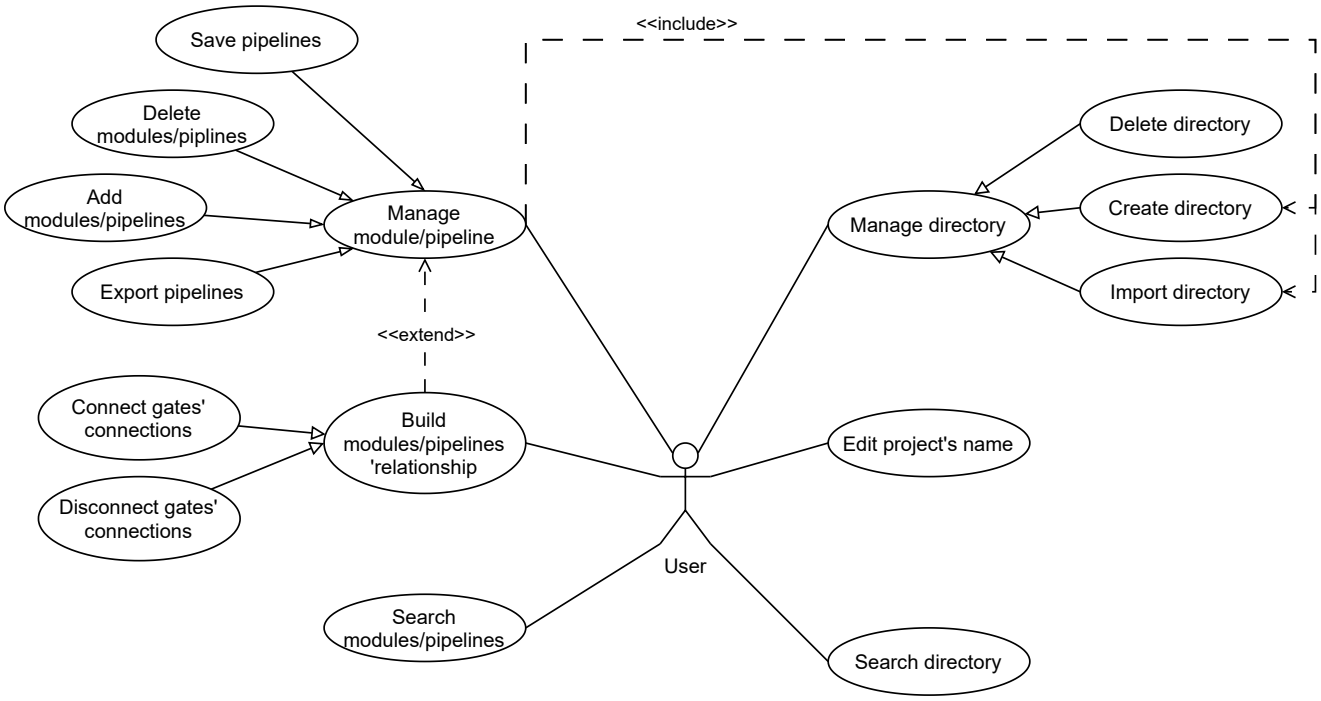
Introduction

AIGENeraTor is an application designed for creating AI or programs in general with drag-and-drop method. This application is currently created for educational purpose, so the perfomance of programs created with this aplication might have quite bad performance.

Problem statement

AI and Machine Learning (ML) have rapidly evolving recently and become a common and useful tool in our daily life. However, these technologies seem to be quite complicated, especially for newbie. We suffer these problems ourselves. Thus, we decide to implement an application to create AI pipelines so that everyone can take use of the basic level of AI as well as visualize them. We hope AIGENT can be a useful tool to make learning, building, and using AI become an easy and enjoyable task for everyone.

Usecase Diagram



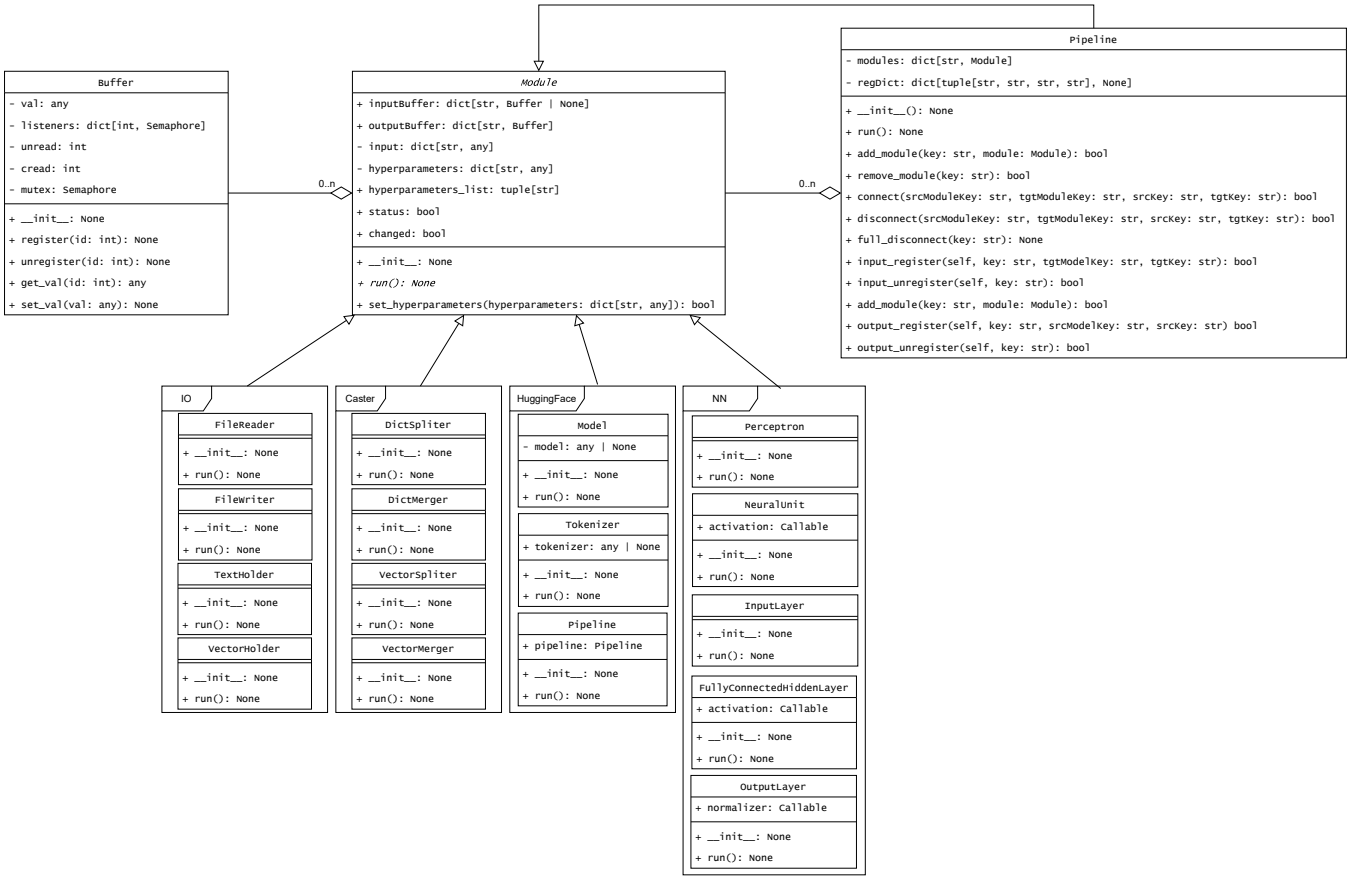
The diagram visually depicts:

User Interactions: How users (actors) can interact with the system to manage pipelines.

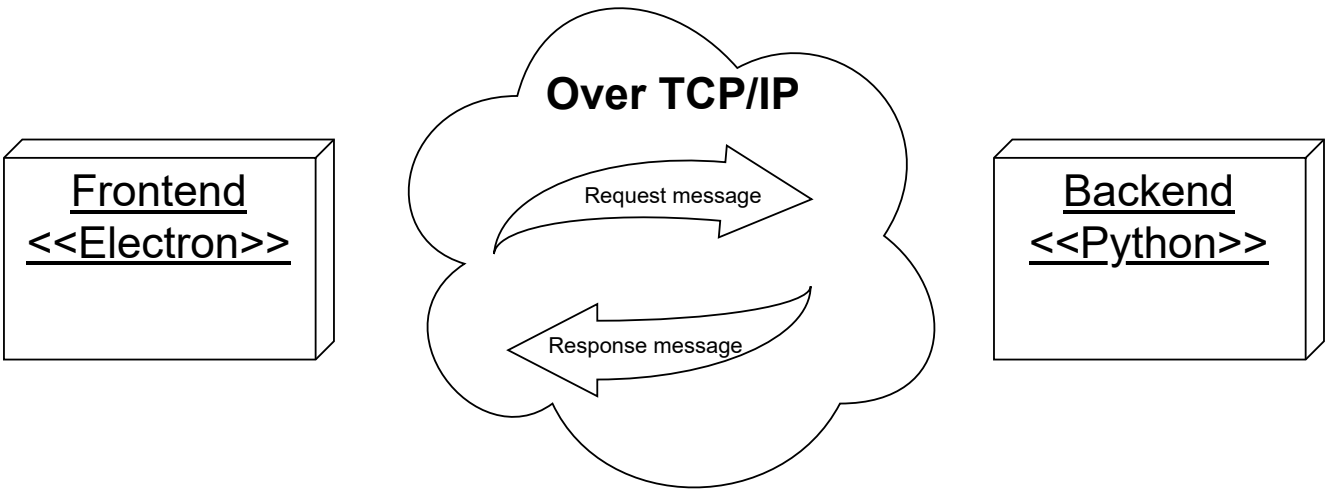
Functionalities: Key functionalities available to users, including creating directories, searching pipelines, managing projects, building/deleting pipelines, and exporting/importing data.

Relationships: Connections between functionalities, such as nested actions and optional extensions. This use case diagram provides a clear understanding of user interaction capabilities within the pipeline management system.

Class Diagram



Architechture



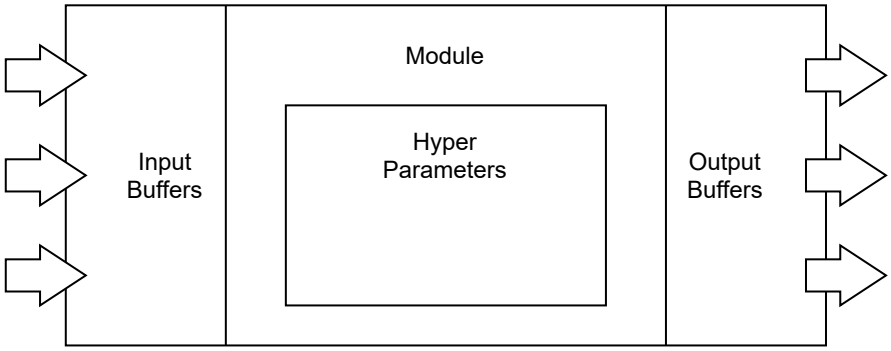
Our program consist of two parts, which can work independently, frontend and backend. While backend is implemented with Python, frontend is coded with Electron, a popular Node.js framework for desktop development that have web like syntax - which not seem to help at all. These two parts communicate with each other purely by json strings sended through TCP/IP sockets.

Backend

To create a drag-and-drop AI pipelines, we split them into modules, which is anything can work independently with some given input and return some output, do something, or both.

Modules

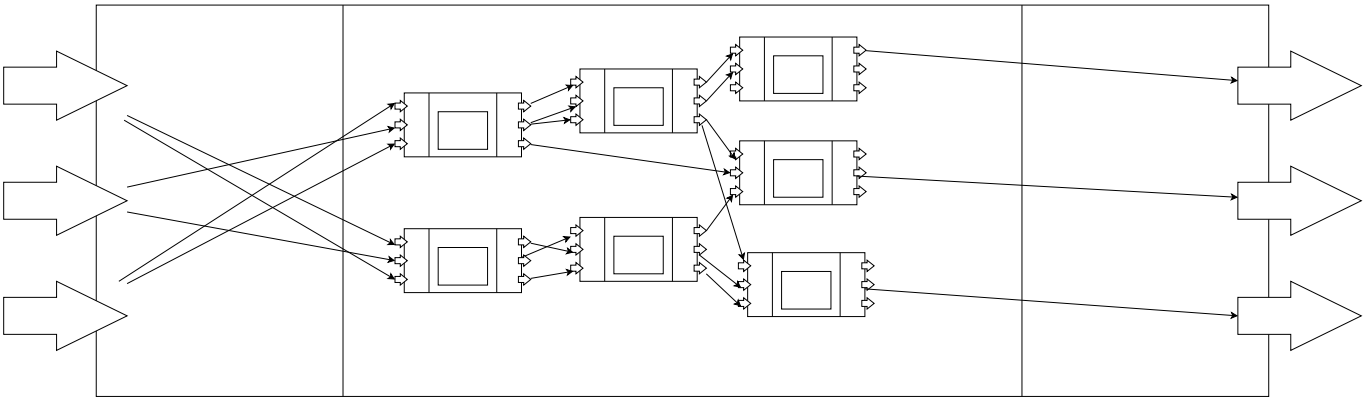
Modules consist of three main components: output buffer, which is designed to hold and maintain value for asynchronous accesses; input buffer, which holds pointers to the correspond source: an output buffer of another module; and a run method, define how a module compute, act, and map input value to output value. This design allows modules to work all by themselves, provides good modularity.



Our backend is designed to allow developers to easily scale built-in modules. Just implement your own module following the implemented modules we provide in to the available files or a new python file and place it in python folder and it will work fine. However, our frontend has not support this feature yet due to difficulties in file reading and text processing.

Pipeline

To make it easier for users to build bigger and bigger applications, we implement a Pipeline as if it is a module. Pipelines inherit all nature of modules with additional information about the modules that the pipeline has. Pipelines run by activating all modules and letting them run in multi-thread. This allows us to maximize the flexibility of the pipeline with a simple design. However, this design heavily affects the performance of the pipeline, especially for feed-forward pipelines consisting of many modules.



A pipeline can be saved, and because pipelines are also modules, it can be added into other pipelines. Hence, our program allows users to build up large pipeline by gradually building smaller pipelines.

Frontend

Our program features a user-friendly frontend comprising two main pages: the Homepage and the Workspace.

Homepage

The Homepage serves as the control center for directories, which contain the user interface, backend, and connections of the modules. On this page, users have access to several essential functions:

Create New Directory: Users can create a new directory to navigate to the Workspace page so as to start organizing their modules and workflows.

Search Directory: A search function allows users to quickly find existing directories.

Delete Directory: Users can remove directories they no longer need.

Import Directory: This special feature enables users to import their directories, provided they adhere to the program's format requirements.

Access Workspace: Users can click on an existing directory to navigate to the Workspace page, where they can edit pipelines, modules, and connections between gates within that directory.

Workspace

The Workspace is where the core module interactions and pipeline creations occur. It features a menu that provides access to both pre-built modules and user-created modules. Key functionalities on this page include:

Drag and Drop Modules: Users can easily drag and drop modules from the menu into the workspace.

Connect Gates: Users can connect the gates of various modules to form a larger, interconnected pipeline.

Set Hyperparameters: For added flexibility, users can click on individual modules to set hyperparameters, tailoring each module's behavior to their specific needs.

Delete Connections and Modules: Within the workspace, users can delete connections between gates and remove modules that are no longer needed, ensuring the workspace remains clean and efficient.

Export and Save: The Workspace also offers functions to export the entire pipeline in a predefined format and save the current state of the pipeline for future use.

Key Features

1. **Drag and Drop Modules:** Our platform offers an intuitive drag-and-drop interface that allows users to seamlessly add modules to their workspace. Users can effortlessly move these modules around the workspace, arranging them in any configuration that suits their workflow.

2. **Zoom and Pan:** To enhance user experience, the workspace supports zooming in and out, providing users with the flexibility to focus on specific modules or get an overview of the entire project. Additionally, users can pan across the workspace to navigate their modules easily.

3. **Flexible Module Positioning:** Users can position modules anywhere within the workspace, giving them complete control over the layout. This feature ensures that users can organize their workspace in the most efficient and personalized manner.

4. **Connectable Gates:** Each module comes equipped with gates that can be connected to gates of other modules. This allows users to create complex workflows by linking modules together to perform

specific tasks or functions. The connectivity between modules enables the execution of customized operations tailored to the user's requirements.

5. **Hyperparameter Setting:** Users can click on modules to set hyperparameters, allowing for precise control and customization of each module's behavior. This feature enables users to fine-tune their modules to achieve optimal performance for their specific tasks.

How to Use the Program

1. Run file **Main.exe** in the folder named **Main**
2. Run file **frontend.txt** in the folder named **frontend ...**

Starting on the Homepage:

1. **First Time Setup:**

Click the "Add New" button to create a new directory. This will take you to the next page.

2. **Editing an Existing Directory:**

If you already have a directory and want to edit it, click on the directory frame to move to the Workspace page.

On the Workspace Page:

1. **Editing Project Name:**

To edit the project name, click the edit icon and enter the new name.

2. **Adding Modules:**

Click and hold a module from the side menu. Drag it to the workspace and release the mouse button to place it.

3. **Zooming and Panning:**

Use the scroll wheel on your mouse to zoom in and out of the workspace.

4. **Connecting Gates:**

Each module has a different number of input and output gates.

To connect gates: Hover the cursor over a gate until it changes to a hand icon. Click and hold the gate, drag to another gate (which will also show a hand icon), and release the mouse button to create the connection.

5. **Deleting Connections and Modules:**

To delete a connection: Right-click on the connection line. To delete a module: Right-click on the module (ensure the hand icon is over the module before right-clicking).

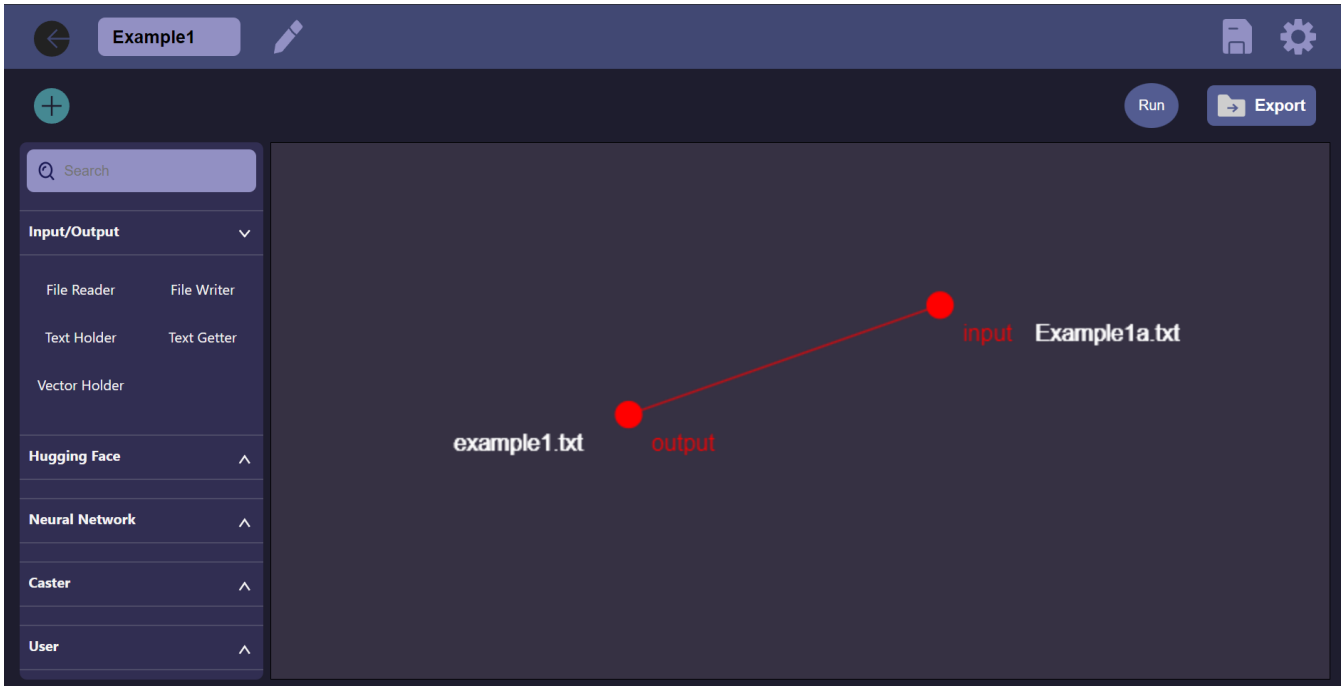
6. **Saving and Running the Project:**

To save your work, click the save icon. To run the project and see the output, click the run button. To export the project as a JSON file, click the export button.

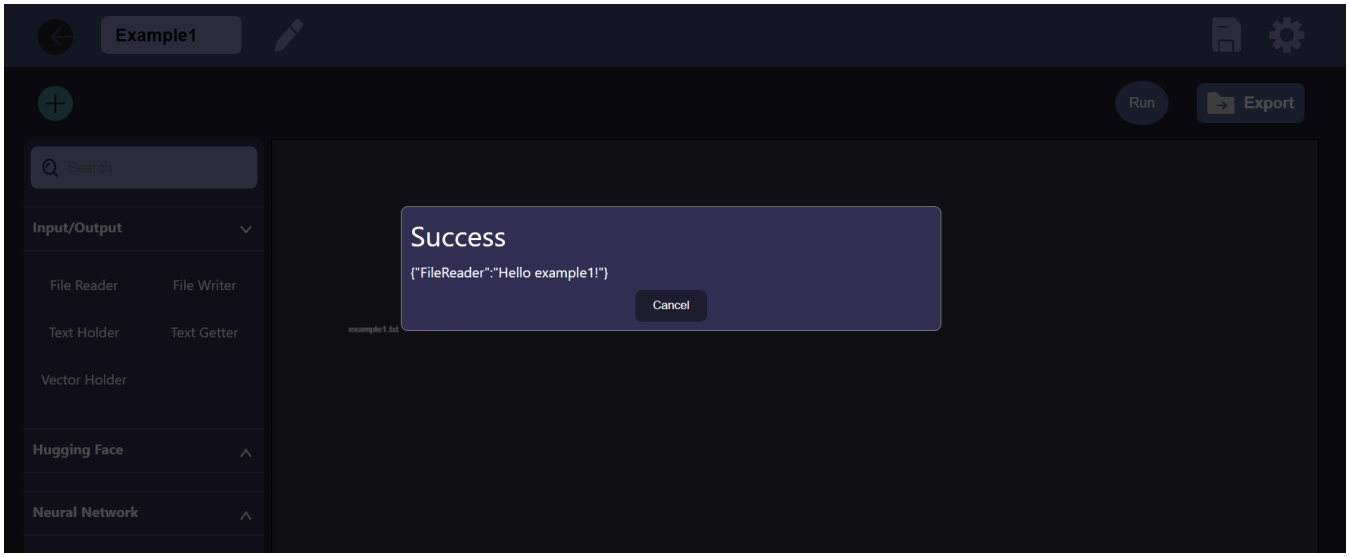
Example 1: Read-write file (easy):

1. **Add a New Directory:**
    - Click the "Add" button on the Homepage to create a new directory and navigate to the Workspace page.
2. **Edit Project Name (Optional):**
    - If you wish to edit the project's name, click the edit icon and enter the new name. If not, you can skip this step.
3. **Add a File Reader Module:**
    - Left-click on the "File Reader" module in the side menu, hold, and drag it into the workspace.
4. **Select a File for the File Reader Module:**
    - Click on the dragged File Reader module to select a file. Choose any file in the .txt format. For this example, use: `Example1.txt` with the content: `Hello example1!`
5. **Add a File Writer Module:**
    - Repeat the same process as in step 3, but this time select the "File Writer" module.
    - Perform the file selection step for the File Writer module. For this example, use an empty file:
        - `Example1a.txt`
6. **Connect the Modules:**
    - Move the cursor to the output gate of the File Reader module and drag it to the input gate of the File Writer module. Note that the module names are not displayed during this process, so remember the module positions.

The final visualization is:



7. **Save the Project (Optional):**
    - Click the Save icon to save your work. This step can be skipped if not needed.
8. **Run the Project:**
    - Click the Run button. If the operation is successful, a success notification will pop up. If there is an error, an error notification will appear.



The file Example1a.txt is updated with content: `Hello example1!`

Example 2: Pipeline hugging face (Medium):

1. **Add the "Text Holder" Module for Question Input:**
    - Drag and drop the **"Text Holder"** module from the **"Input/Output"** section or use the search function to locate it.
    - Click on the module to input the text, which should be the question. In this step, input: `What is the deadline?`
2. **Add the "Text Holder" Module for Context Input:**
    - Repeat step 1 to add another **"Text Holder"** module.
    - Click on this module to input the text, which should contain the key answer. In this step, input: `The food is 5. The deadline is August.`
3. **Add the "Dict Merger" Module:**
    - Drag and drop the **"Dict Merger"** module from the **"Caster"** section.
    - Click to set the hyperparameters and input: `question,context`. After inputting, you may need to zoom in or out to refresh the display.
4. **Add the "Pipeline" Module:**
    - Drag and drop the **"Pipeline"** module from the **"Hugging Face"** section.
    - Set the following parameters:
        - **task:** `question-answering`

        - **model:** `deepset/tinyroberta-squad2`
        - **config:** `deepset/tinyroberta-squad2`
    - Note: You can use most models available on Hugging Face.

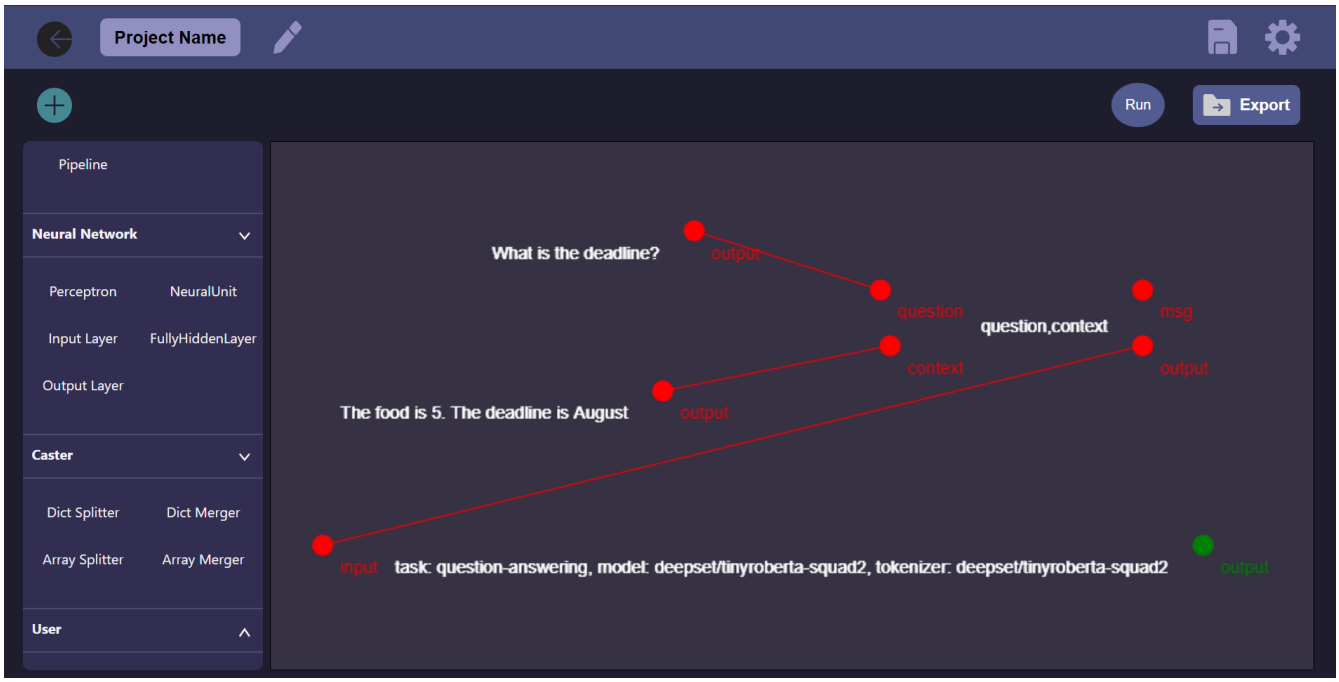
5. **Connect the Gates:**

    - First, connect the output gate of the **Text Holder** (containing the question) to the question gate of the **Dict Merger** (content: question, context).
    - Second, connect the output gate of the **Text Holder** (containing the key answer) to the context gate of the **Dict Merger**.
    - Third, connect the output gate of the **Dict Merger** to the input gate of the **Pipeline**.

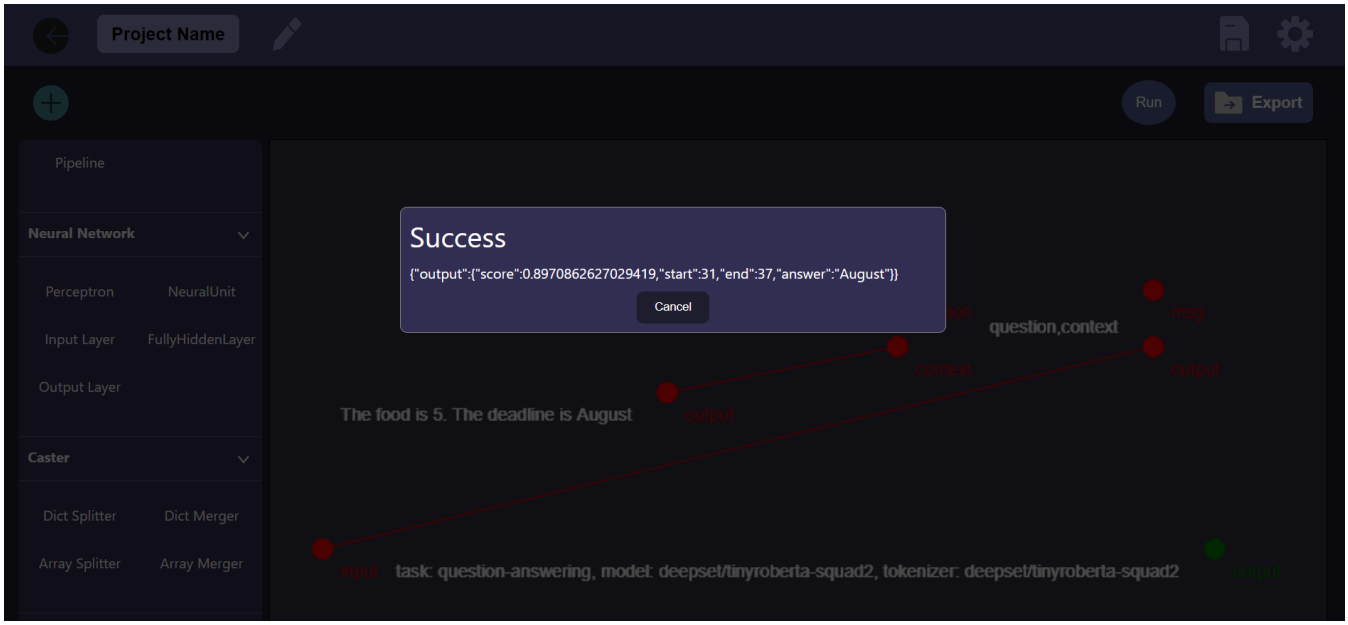
6. **Register the Output:**

    - Double-click the output gate of the **Pipeline** module and input: `output` to register the output.

The final visualization is:



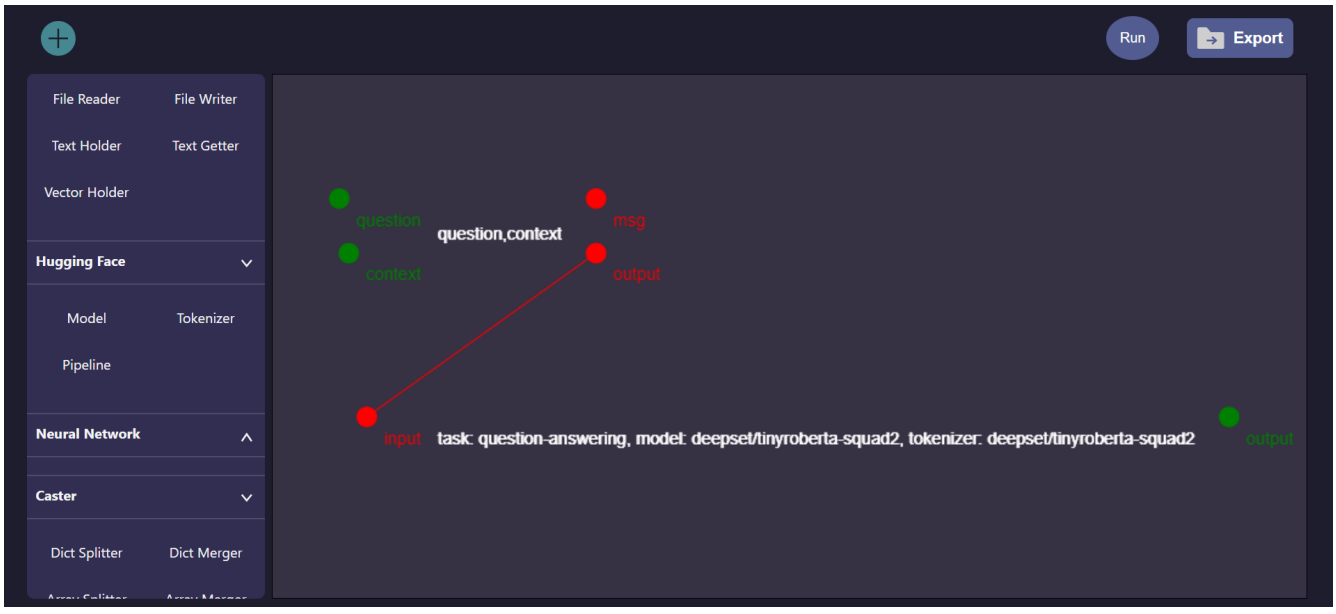
7. **Run the Project:**
    - Click the Run button.



Example 3: New pipeline using the user's module (Medium):

1. To proceed, ensure that at least one directory has been created. In this example, we utilize the Hugging Face pipeline example 2. However, there's a slight variation: we won't add the TextHolder module. Instead, we'll register inputs for both input gates of the Dict Merger module by double clicking. The question gate, we input: `question` and the context gate, we input: `context`.

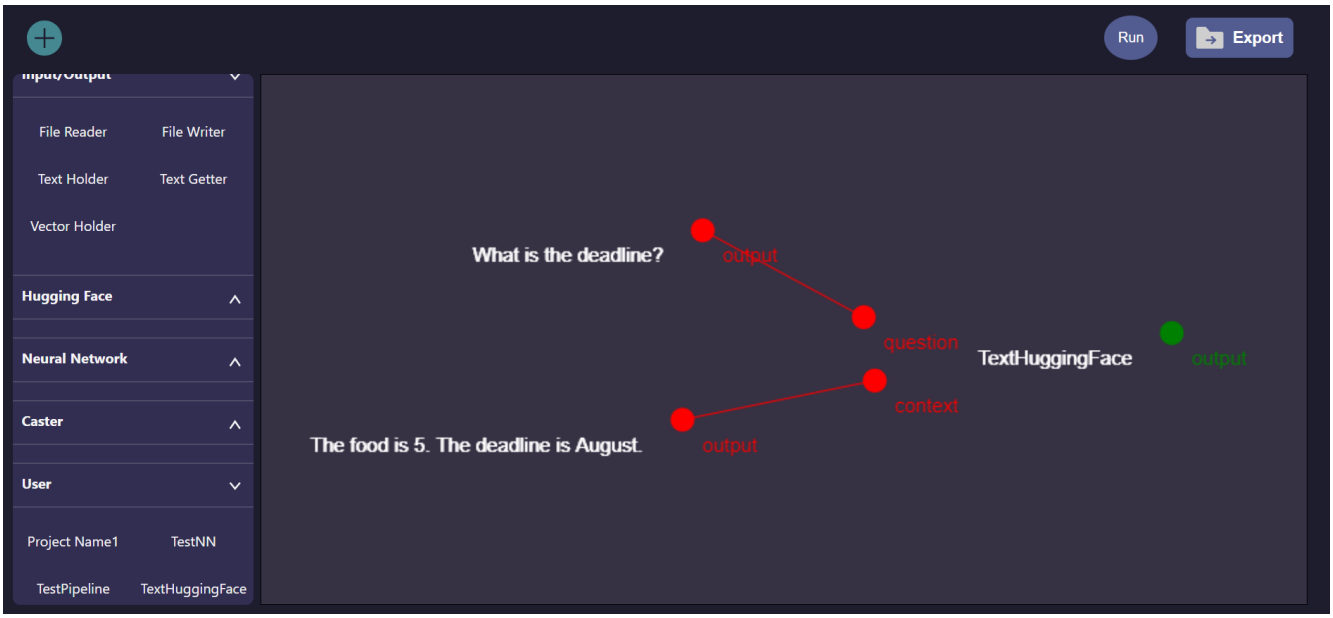
The visualization of that directory is:



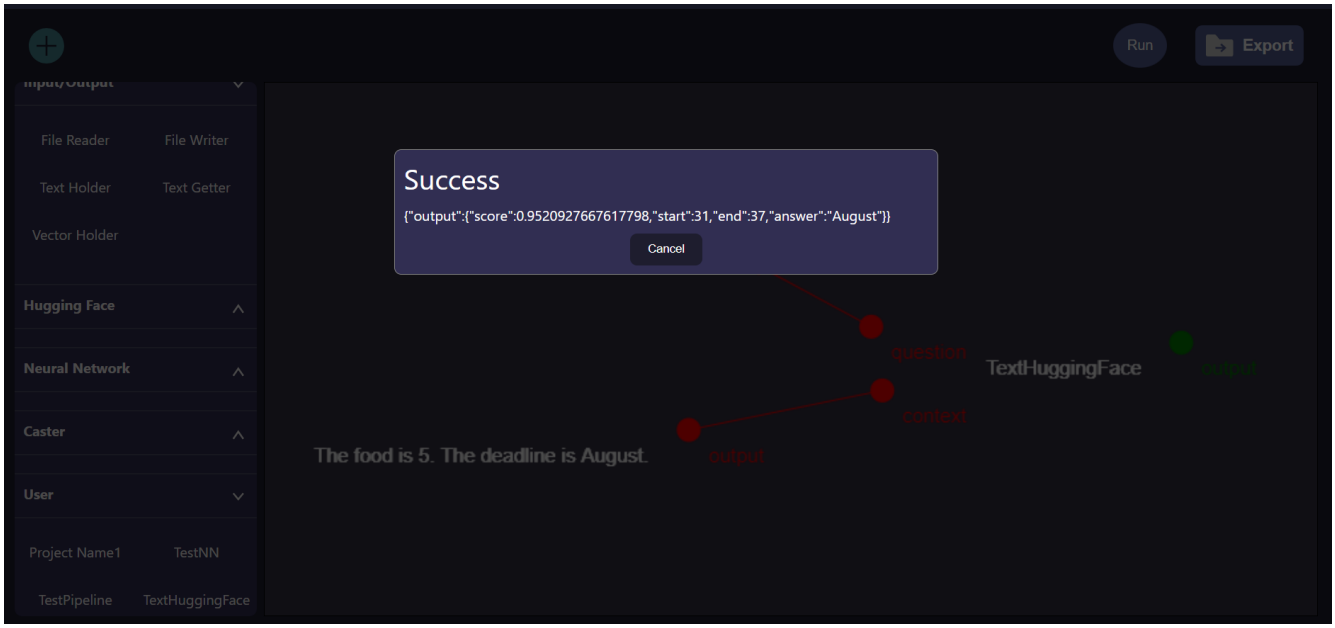
2. **Choose or Create a Directory:**
    - You can either create a new directory or access an existing one. In this example, we'll create a new directory.
        - **A.** Follow steps 1 and 2 of example 2.
        - **B.** Drag and drop your created modules from the User section. For instance, drag and drop the module `TextHuggingFace` created in step 1 above.
        - **C.** Connect the gates as follows:

    - Connecting the output gate of TextHolder (containing the question) to the question gate of the module TextHuggingFace.
    - Connecting the output gate of TextHolder (containing the key answer) to the context gate of the module TextHuggingFace.
- **D.** Register the output by double-clicking on the output gate of the module TextHuggingFace.

The visualization is:



- **E.** Click "Run" button. It returns to output: `August`.



Example 4: Build XOR gate using neural network (Hard):

1. **Add the "Vector Holder" Module:**
    - Drag and drop the module "Vector Holder" from the Input/Output section.
    - Click to set: `1,1`

2. **Add the "NeuralUnit" Module (First Configuration):**

    - Drag and drop the module "NeuralUnit" from the Neural Network section.
    - Click to set:
        - weights: `1,1`
        - bias: `0`
        - activation: `ReLU`

3. **Add the "NeuralUnit" Module (Second Configuration):**

    - Repeat step 2, but with different settings:
        - weights: `1,1`
        - bias: `-1`
        - activation: `ReLU`

4. **Add the "Array Merger" Module:**

    - Drag and drop the module "Array Merger" from the Caster section.
    - Click to set `2`.

5. **Add the "NeuralUnit" Module (Third Configuration):**

    - Repeat step 2, but with different settings:
        - weights: `1,-2`
        - bias: `0`
        - activation: `ReLU`

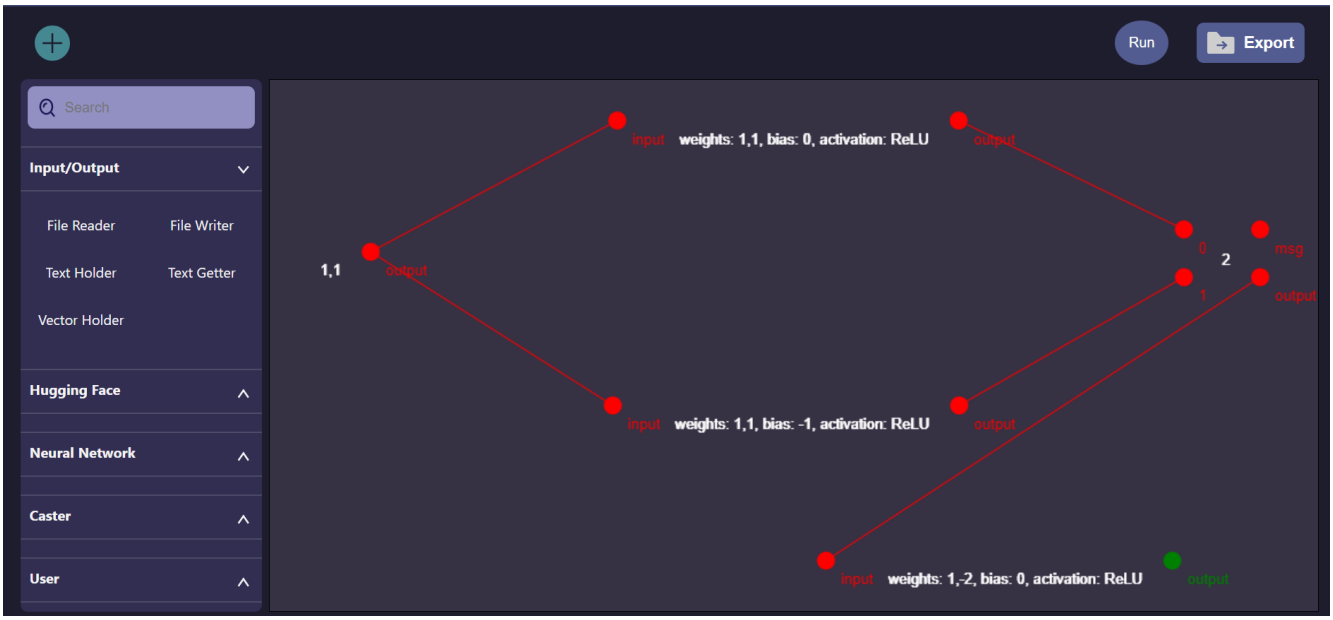
6. **Connect the Gates:**

    - Connecting the output gate of Vector Holder to the input gate of NeuralUnit (weights: 1,1).
    - Connecting the output gate of NeuralUnit (weights: 1,1, bias: 0) to the `0` gate of Array Merger.
    - Connecting the output gate of NeuralUnit (weights: 1,1, bias: -1) to the `1` gate of Array Merger.
    - Connecting the output gate of Array Merger to the input gate of NeuralUnit (weights: 1,-2).

7. **Register the Output:**

    - Register the output gate of NeuralUnit (weights: 1,-2) by double-clicking the output gate and set to `output`.

The final visualization is:



8. **Run the Project:**
    - Click the Run button.