Is it an HPC Workflow Assistant? Is it a Framework? It's Drona Workflow Engine

Andrii Kryvenko
Texas A&M University
College Station, TX, USA
andriikryvenko@tamu.edu

Duy Pham
Texas A&M University
College Station, TX, USA
duy@tamu.edu

Marinus Pennings
Texas A&M University
College Station, TX, USA
pennings@tamu.edu

Honggao Liu
Texas A&M University
College Station, TX, USA
honggao@tamu.edu

Abstract

HPC resources are becoming increasingly complex, while HPC itself is becoming more popular among novice researchers across a wide range of research domains. These novice researchers often lack typical HPC skills, which results in a steep learning curve that leads to frustration and inefficient use of HPC resources. To address this, we developed Drona Workflow Engine. Drona offers an intuitive Graphical User Interface (GUI) that assists researchers in running their scientific workflows. The researcher provides the required information for their specific scientific workflow, and Drona generates all the scripts needed to run that workflow on the researcher's behalf. For transparency and additional flexibility, Drona will display all generated scripts in a fully editable preview window, allowing the researcher to make any final adjustments as needed. Drona also provides a flexible framework for importing, creating, adapting, and sharing custom scientific workflows. Drona significantly enhances researcher productivity by abstracting the underlying HPC complexities while retaining full control over their workflows.

CCS Concepts

• **Software and its engineering** → *General programming languages*; **Frameworks**; • **Human-centered computing** → *User interface design*; **Graphical user interfaces**.

Keywords

High Performance Computing, Frameworks, Scientific Workflows, Batch processing, Schedulers, GUI

ACM Reference Format:
Andrii Kryvenko, Duy Pham, Marinus Pennings, and Honggao Liu. 2025. Is it an HPC Workflow Assistant? Is it a Framework? It's Drona Workflow Engine. In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25), November 16–21, 2025, St Louis, MO, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3731599.3767431

ACM ISBN 979-8-4007-1871-7/25/11
https://doi.org/10.1145/3731599.3767431

1 Introduction

HPC resources are becoming increasingly complex, with new types of accelerators, containerized applications, and Kubernetes, among others. At the same time, HPC is becoming increasingly popular with novice researchers who lack the typical HPC skills needed to run scientific workflows. While HPC centers offer training, documentation, and even video tutorials to address these challenges, such resources require a significant time investment and effort from researchers and often only provide a basic understanding of various HPC concepts and procedures. This creates opportunity costs, as researchers must divert valuable time from their actual research. Additionally, HPC helpdesks regularly receive tickets from researchers experiencing issues with running their workflows, leading to frustration for the researcher and placing extra burden on often understaffed helpdesk teams. In general, HPC researchers, whether novice or experienced, should not be hindered by HPC-specific details when conducting their research. To achieve this, we developed Drona Workflow Engine.

Drona is a user-friendly 100% GUI tool that guides the researcher in providing all required information for their scientific workflow, analyzes the provided input for potential issues, generates one or more scripts representing the scientific workflow, and displays the generated scripts in a preview window where the researcher can inspect the generated scripts and make final adjustments if needed. Drona also provides feedback, with varying severity levels, based on the input provided by the researcher. Finally, Drona submits the job on the researcher's behalf. Beyond this, Drona also provides a powerful framework for researchers and system administrators to create, customize, and share new workflows in a largely declarative manner.

The design goals of Drona are as follows:

- Allow researchers to focus on their research
- Allow researchers to be in control of their scientific workflows (including creating, managing, and sharing)
- Ease the burden on HPC helpdesks and system administrators.

Section 3 introduces the Drona GUI interface, showing a step-by-step description from selecting the scientific workflow to actually running it. In Drona terminology, a scientific workflow is represented by a Drona environment. Sections 4 and 5 discuss the structure of Drona environments in great detail and how researchers can create and adjust their Drona environments. Section 6 discusses how Drona manages Drona environments and options for researchers to share their custom Drona environments. Finally, Section 8 explores additional applications of Drona beyond its primary role in scientific workflow generation.

2 Related Work

A significant amount of work has been done to ease the burden on researchers in running their simulations on HPC Clusters. OpenOnDemand [6] is a popular web interface to access HPC cluster that includes a job composer. Their job composer provides batch job templates that the researcher can use. However, it is still mostly text-based. Pham, et. al [14] developed a GUI Dashboard that included a Job Composer where the researcher can enter information in a GUI form. Similarly, OpenComposer [10] provides a form that allows researchers to enter job-specific information. Both Pham, et. al and OpenComposer provided functionality to create fully formed jobs for specific software environments, such as Matlab and OpenFoam. One limitation of all these tools is that they employ a mostly variable substitution scheme to map researcher-provided information in the form directly to placeholders in the templated job scripts: the researcher enters a value in the form, the tool substitutes a placeholder in the job script with that value. This significantly limits functionality. In addition, none of these tools provides advanced feedback capabilities to assist the researcher in providing correct and meaningful information in the form. The Drona Workflow Engine is designed to address these limitations through an advanced mapping framework and dynamic feedback capabilities.

3 Drona: an HPC Workflow Assistant

In this work, we define scientific workflows as coordinated computational tasks that accomplish specific research objectives, ranging from single sophisticated jobs with complex configurations to multi-job sequences with dependencies executed across diverse HPC environments. In Drona, these workflows are represented by drona environments — collections of specification files that define how to generate and execute the computational workflow.

To work with these environments, Drona offers an intuitive Graphical User Interface (GUI) that guides researchers in providing all the relevant and necessary information for their specific workflow. Based on the provided information, Drona will generate one or more scripts and display them in a fully editable preview window. This enables complete transparency into what Drona generated, allowing the researcher to verify the generated scripts and make adjustments as needed. Finally, Drona will run the generated workflow on the researcher's behalf. In the following subsections, we will give a step-by-step explanation of how a researcher executes a workflow from selecting the scientific workflow to actually running the generated workflow.

We will use AlphaFold [8] [5] as a guiding example of a workflow. AlphaFold is an artificial intelligence (AI) program developed by Google DeepMind, which utilizes deep learning techniques to predict protein structures.

3.1 Selecting Environments

The first step in generating a scientific workflow is selecting the actual workflow. Drona uses the concept of an environment to encapsulate a scientific workflow. An environment consists of all the specification files needed to generate the workflow. Sections 4 and 5 will provide a detailed discussion of the internals of Drona environments. In the Drona GUI, the researcher can select a workflow from a dropdown menu. Figure 1 shows a screenshot of the environment-select dropdown with a sample of available environments.

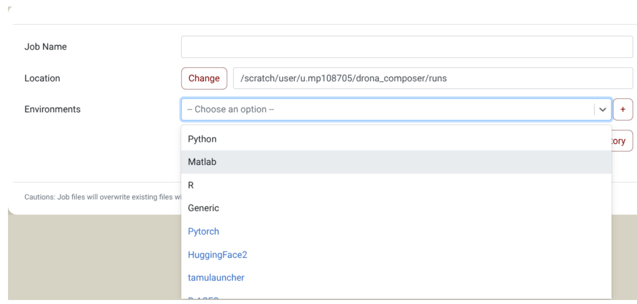

Figure 1: Screenshot of the Drona environment-select dropdown, showing the available environments the researcher can pick from.

The screenshot shows *system* environments in black and *user* environments in blue. System environments are visible to everyone, while user environments are only visible to the researcher. One of Drona's goals mentioned in the introduction is *"to allow researchers to be in control of their scientific workflows"*. This is accomplished by storing environments in the researcher's directory. The researcher has full control over the environment and can adjust it to meet their specific needs. An additional benefit is that the Drona GUI environment-select dropdown will not be cluttered with a large number of scientific workflows the researcher will never use.

3.1.1 *Importing workflows.*

Drona's approach to store environments in the researcher's directory is through the *import* functionality. As seen in Figure 1, there is a "+" button to the right of the environment-select dropdown. Clicking this "+" button, will pop up an import window. Figure 2 shows a screenshot of the import window.

The figure shows a sample of environments that can be imported. The screenshot also shows options to search for specific environments and/or filter out environments, based on the field of science or organization. When the researcher clicks the *Add* button. Drona will copy the environment into the researcher's directory

3.2 Form Window

Once the researcher selects the environment of choice, Drona will populate the GUI form with input fields specific to that particular scientific workflow, guiding the researcher to provide all relevant information needed to generate the workflow. Drona offers a comprehensive set of input elements to make the form as convenient and intuitive as possible for the researcher. Section 4.1 will discuss

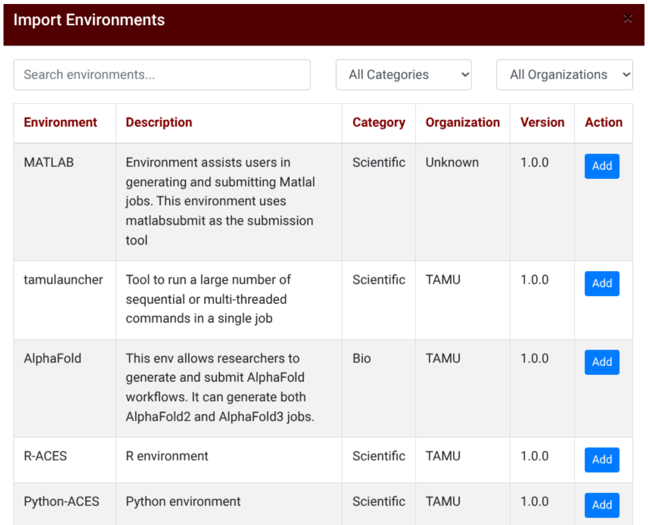

Figure 2: Screenshot of the Drona import windows, showing a subset of available environments with additional information. The top of the windows shows filter options. Researchers can click the "Add" button to import the environment.

form elements in more detail. Figure 3 shows a screenshot of the AlphaFold environment form.

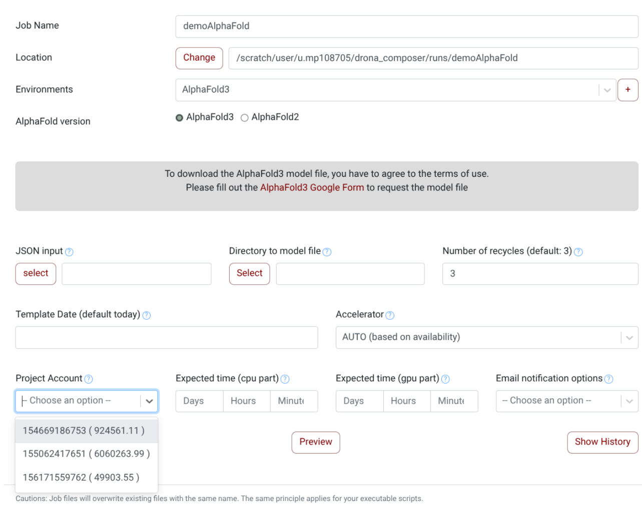

Figure 3: Screenshot of an AlphaFold GUI form showing workflow-specific form fields as well as some optional HPC fields such as the account to use. Drona will pick default values when the researcher doesn't provide a value for a form field.

As shown in the figure, most of the fields are domain-specific. The researcher has the option to use AlphaFold2 or AlphaFold3. In the screenshot, we selected to generate an AlphaFold3 workflow. Most of the fields shown next are specific for AlphaFold3. If the researcher had chosen AlphaFold2, the form would show completely different fields, targeted for AlphaFold2, such as fields to select the FASTA file, the Model Preset, and the DB preset. The figure also displays a dropdown menu that allows the user to select the account to use. Using a dropdown avoids the issue of entering an incorrect account number by mistake. In general, if the researcher doesn't provide information for some of the fields, Drona will either use default values or deduce based on values provided for other fields. Note the accelerator field, which indicates that the researcher has selected "AUTO". In this case, Drona will choose the type of accelerator automatically, based on availability and projected start times (and potentially taking into consideration performance differences between the various accelerators). A common problem for many HPC clusters is the researcher's demand for accelerators. From the researcher's perspective, letting Drona select the accelerator means that their scientific workflows might start executing faster. Additionally, this can also help with distributing jobs more evenly across accelerators.

3.3 Preview Window

Once the researcher provides the necessary information, Drona will generate one or multiple scripts using an advanced mapping scheme. Section 5 will provide a detailed discussion of the mapping step. During the mapping step, the Drona environment can also analyze the provided information for correctness and efficiency. Drona will then display the generated scripts in an editable preview window with full syntax highlighting, allowing the researcher to verify the scripts and, if necessary, make any last-minute adjustments. In addition to showing the generated scripts, Drona can also provide immediate feedback through various types of messages shown in the preview window.

Figure 4 shows the preview window for the AlphaFold workflow. The preview window is divided into two parts. The left side shows the editor pane with tabs for all generated scripts, and the right pane shows the output pane. Once the researcher runs the workflow, the output will be shown in the output pane.

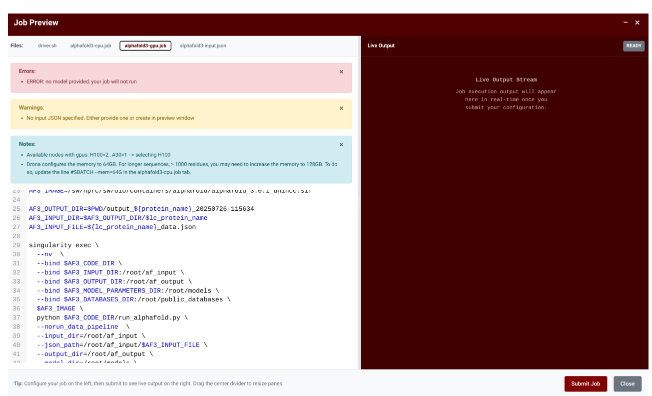

Figure 4: Screenshot showing a preview window with generated scripts in separate tabs for an AlphaFold3 environment. Also showing feedback messages of various severity levels.

Figure 4 also shows various color-coded messages: an error (red background), a warning (yellow background), or a note (blue background). For illustration purposes, we intentionally left out needed information in the form to show the feedback capabilities of Drona. The note (blue background), indicates that Drona selected an H100 GPU based on availability.

Once the researcher clicks the "Submit Job" button, Drona will run the workflow and the ouput pane will show the output. In the case of AlphaFold, Drona will submit two independent Slurm jobs: a CPU job and a GPU job that depends on the successful completion of the CPU job. The output window will display Slurm information for both submitted jobs. Section 4.2.1 will discuss in more detail how workflows are run.

3.4 Reproducibility

Reproducibility is an important aspect in running scientific workflows. Every job launched through Drona has its complete context, including all provided inputs and the final generated files, stored in a database. The Drona GUI offers multiple options for researchers to inspect past runs and launch precise replicas of any previous experiment with a single click. One way to do this is through the history section in the Drona GUI, which is shown in Figure 5.

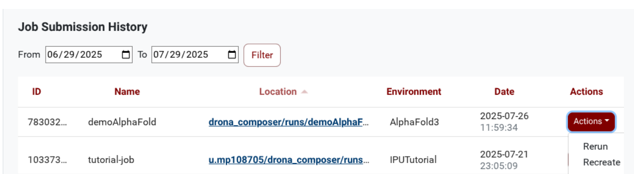

Figure 5: Screenshot showing Rerun/Recreate functionality to rerun or recreate a previously run workflow

The figure 5 shows two recently submitted jobs. The "Actions" dropdown in the last column shows two options to rerun a previously executed workflow: rerun and recreate.

- **Recreate jobs**. This option automatically selects the environment that matches the job and pre-populates all fields with the values used in the previously run job. This option will generate the exact same workflow as the previous job. The re-create option will NOT preserve any changes made in the preview editor window.
- **Rerun jobs** This option will bypass the form and immediately show the preview window with the exact contents of the previously run job, including any changes made in the preview editor window.

4 Drona: a Workflow Framework

The Drona Workflow Framework allows researchers to create and adapt workflows in a mostly declarative manner. In the previous section, we demonstrated how researchers can import environments and generate workflows. In this section, we will discuss the structure of an environment in more detail. Figure 6 shows an overview of a Drona environment and how the Drona Framework generates workflows from environment specifications.

The figure shows the specification files used to create the form and generate the final workflows. Every Drona environment should contain the following specification files.

- file schema.json, describing the elements in the form
- file map.json, describing a mapping from input values, provided by the researcher, to placeholder values used in the driver and template files
- file driver.sh, the main script that starts the workflow execution
- template files, templates needed to create the workflows with placeholders that will be replaced with final string values

In Section 4.1, we will describe in detail the specification file `schema.json` and the available form elements. In Section 4.2, we will discuss in detail the specifications of the driver and templates. The core of the Drona framework is the advanced mapping capabilities, and we will discuss the mapping step in Section 5

4.1 Form Elements

Figure 3 showed a small subset of the available form elements, such as number fields, file selectors, radio button groups, and dropdowns. Beyond these examples,Drona provides a large, growing set of form elements, which are used in the JSON-formatted file `schema.json` that contains the information Drona needs to render the form

Every form element in `schema.json` has the following fields:

```
{
  "<NAME>": {
    "type": "<ELEMTYPE>",
    "label": "<LABEL>",
    "name": "<VARNAME>",
    "help": "<HELPMESSAGE>",
    "condition": "<LOGICAL CONDITION>"
  }
}
```

where **type** indicates the type of element (), **label** is the label for the element. **name** is the variable name associated with the value the researcher provides for this element, **help** is an optional field to provide additional information to the researcher, and **condition** field adds a logical condition to determine whether the element will be displayed. Section 4.1.1 will provide more details about the various element types, and Section 4.1.2 will discuss conditionals in more detail.

These are the basic fields that are shared across all form elements. Depending on the element type, additional element fields can become available.

4.1.1 Available form element types.

Drona provides a rich and easily expandable library of form elements. Basic elements include number and text fields, while radioGroup and select elements offer standard choice functionality with hardcoded options. More advanced elements include time and unit fields for batch-scheduler specifications, and a module element that provides a powerful search interface for finding and adding software modules. File handling is supported through uploader and picker elements that allow researchers to upload and select files.

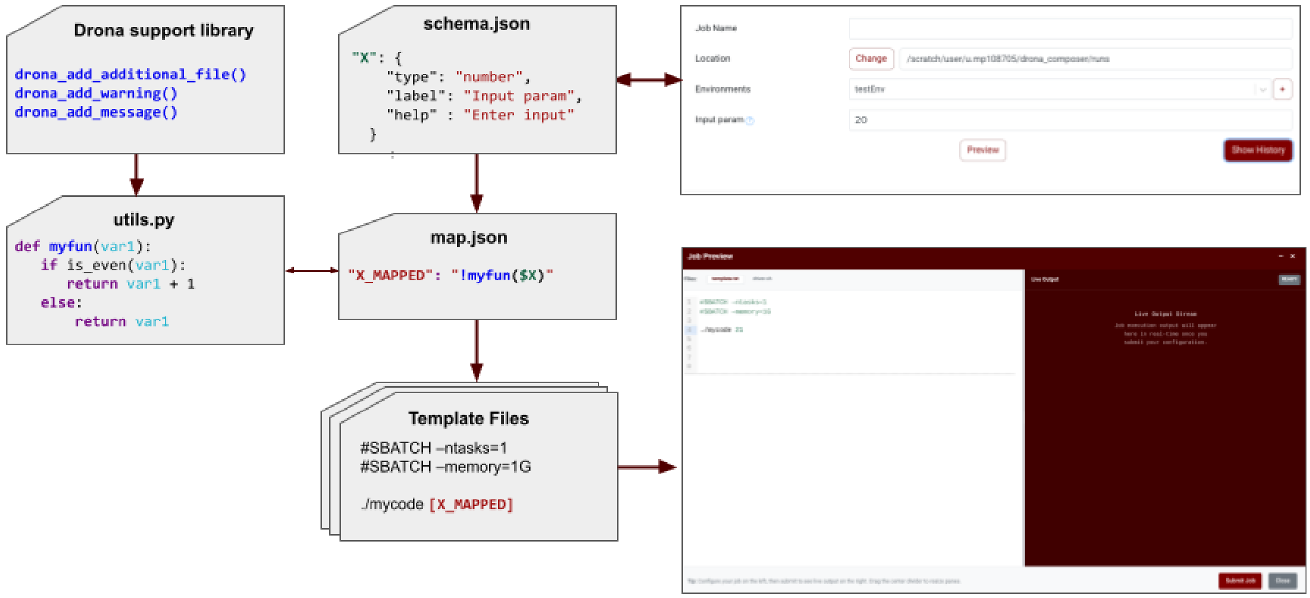

Figure 6: The Drona Workflow Engine overview demonstrating the complete workflow from researcher input to executable job generation. Researchers interact with a dynamic web form whose structure is defined by a declarative `schema.json` file. When researchers submit the form, Drona uses `map.json` file to process the inputs and, if needed, invokes functions from `utils.py` to generate optimized job scripts and configurations. Functions in `utils.py` have access to the Drona support library for additional functionality.

The dynamicSelect element extends standard selection functionality by executing scripts to retrieve options dynamically, while the StaticText element can render HTML content and refresh by executing scripts to fetch updated content. For a complete and up-to-date list of available elements, please refer to the Drona Form Elements page [17].

4.1.2 *Conditionals.*

The "condition" field field enables dynamic form behavior by controlling when elements are displayed based on other form inputs. The basic syntax of a condition field is as follows:

$$"condition" : " < ElementName > .value1 \tag{1}$$

Where **<ElementName>** is the element's name and **value1** is the value entered by the researcher. <ElementName> can be any of the basic elements: select, dynamicSelect, radioGroup, checkbox, number, text. Complex logical expressions are also supported using standard operators: && (and), || (or), ! (negate), and parentheses for precedence control, allowing for sophisticated conditional display logic.

4.1.3 *Sample Form JSON.*

The code below shows a snippet of the AlphaFold `schema.json` file. The snippet shows two Form elements: a picker element and dynamicSelect element

```
{
  "JSON": {
    "type": "picker",
    "label": "JSON input",
    "name": "jsoninput",
    "help":"Select JSON input file",
    "localLabel": "select",
    "showFiles": "True",
    "files": "True",
    "defaultPaths": {
        "HOME": "$HOME",
        "SCRATCH": "/scratch/user/$USER",
        "Examples" : "/data/alphafold3/examples/"
    },
    "useHPCDefaultPaths": "False",
    "condition": "alphafoldversion.alphafold3"
  },

  "account": {
    "type": "dynamicSelect",
    "label": "Project Account",
    "help": "provide the account number",
    "name": "account",
    "retriever" : "list_accounts"
  }
}
```

Both elements contain several additional fields. Note the condition field for the **JSON** element. The picker element will only be displayed when the researcher selects AlphaFold3 for the AlphaFold version. The **account** element is a dynamicSelect dropdown, and its values will be retrieved dynamically by executing the *list_accounts* script.

4.2 Driver and Templates

The mapping step, discussed next in Section 5, maps the researcher-provided inputs to placeholder strings. After that, Drona will iterate over the driver and templates, replacing all placeholders with actual string values. In Drona, any text enclosed with [] is considered a placeholder

4.2.1 The Driver.
File `driver.sh` is a required file for any Drona environment. The purpose of the driver is to start the workflow. Below is a basic driver.sh file that submits the workflow to the Slurm batch scheduler. Note, [generatedalphafold] and [flocation] are placeholders, and Drona will replace these with the actual string values retrieved in the mapping step.

```
#!/bin/bash
source /etc/profile

cd [flocation]
/sw/local/bin/sbatch [generatedalphafold]
```

The above sample is a basic yet common case where the driver invokes Slurm to submit the workflow as a batch job. In practice, the driver can run a workflow in any way. In Section 3.3, we saw that the actual AlphaFold driver submited two Slurm jobs. The driver can as easily consist of a pipeline, submitting a series of jobs. Instead of submitting the workflow to the Slurm scheduler, the driver can submit it to a Kubernetes cluster, SSH to a compute node directly in the shell, or utilize middleware APIs, such as TAPIS [15], to start the workflow on another cluster.

4.2.2 Template Files.
Templates describe most of the workflows and can include batch scripts (e.g., Slurm scripts), code files (e.g., Python scripts), configuration files, or any other files used in scientific workflows (e.g., README files, acknowledgement info). Drona provides two ways to specify the list of template files to include: static or dynamic. In Section 5.3.2, we will discuss how to add template files dynamically. Here we will discuss how to add templates statically.

Drona will check for a file named *additional_files.json* with the following structure:

```
{
   "file_name": "<filename>",
   "preview_name": "<preview name>",
   "position": <index>
}
```

where *file_name* is the path, either relative to the environment directory or absolute, *preview_name* is the name shown in the preview tab, and *position* is the tab position (-1 indicates to exclude the file from the preview).

4.2.3 Example Template file.
Below is a snippet of the GPU part of the AlphaFold3 workflow, with some of the code removed for brevity. The snippet displays placeholders [gpu], [gputime], [ALPHAFOLD3MODEL], [timestamp], [maxtemplate], [RECYCLES], and [OLDGPUFLAGS] (where OLDGPUFLAGS represent additional parameters required for certain older GPU types). Drona will replace all these placeholders with string values retrieved in the mapping step.

```
#!/bin/bash

#SBATCH --job-name=alphafold3-gpu
#SBATCH --time=[gputime] --mem=64G [gpu]
#SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=8

module purge
module load AlphaFold3/3.0.1

AF3_MODEL_PARAMETERS_DIR=[ALPHAFOLD3MODEL]

AF3_OUTPUT_DIR=$PWD/output_${protein_name}_[timestamp]

singularity exec \
  --db_dir=/root/public_databases \
  --max_template_date="[maxtemplate]" \
  --num_recycles=[RECYCLES] [OLDGPUFLAGS]
```

5 Mapping

Drona employs an advanced intermediate mapping scheme to generate placeholders from researcher-provided input values. Figure 7 shows a graphical representation of the approach Drona uses. This intermediate mapping step enables Drona to generate complex and advanced workflows, comprising multiple workflow files and dynamic drivers. This also enables the verification and correction of input values, the addition of advanced feedback capabilities, and the integration of command-line tools into a Drona environment.

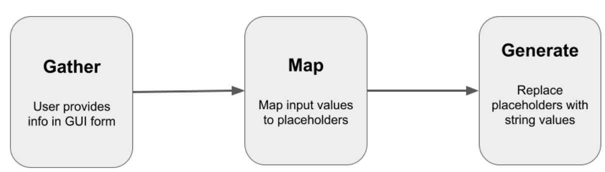

Figure 7: This figure shows the gather, map, generate approach in Drona, with the intermediate mapping step.

5.1 The Mapping File

The mappings are defined in a file named *map.json*. This JSON-formatted file contains a list of mappings, where each element can be defined using the following regular expression:

$$placeholder \leftarrow [STRING\_ELEM]^{+} \tag{2}$$

Where STRING_ELEM is one of the following:

- **variable**: syntax $variable
- **function call**: syntax !myfun($var1, ... , $varN)
- **string literal**: any string that is not a variable or function call

Drona Engine will parse file *map.json* to create an updated mapping with $placeholder \leftarrow stringvalue$. Algorithm 1 shows pseudocode for the mapping step.

Algorithm 1: Mapping Placeholders

```
Input: map.json, inputs.map
Output: Modified map.json
for each (lhs, rhs) in map.json do
    line ← split(rhs, ' ') new_rhs ← '' ;
    for each elem in line do
        if elem starts with $ then
            new_rhs ← new_rhs + inputs.map[elem] + " "
        end
        else if elem starts with ! then
            parse input parameters for elem
             new_rhs ← new_rhs + eval(elem) + " "
        end
        else
            new_rhs ← new_rhs + elem + " "
        end
    end
    replace (lhs, rhs) in map.json with (lhs, new_rhs)
end
return map.json
```

5.1.1 Mapping Example.

The code below shows a snippet of the *map.json* file for the AlphaFold environment with two elements. The right-hand side (rhs) on the first line shows a simple variable substitution denoted by the $ marker. The rhs on the second line shows a function call, denoted by the ! marker.

```
{
   "flocation": "$location",
 "maxtemplate": "!retrieve_maxtemplate($maxtemplate)",
}
```

5.2 The utils.py File

The previous section showed that a mapping can include calls to Python functions. These functions are defined in file *utils.py*. These Python functions are highly flexible and can perform advanced analysis of input combinations, dynamically adjust input values, and incorporate automation, such as selecting the most suitable accelerator. For example, Nasari et al. [11] demonstrated that certain types of accelerators may perform better depending on the batch size. A Drona environment can leverage this information and automatically decide the optimal accelerator to use.

5.2.1 utils.py Example .

The previous section showed a snippet of the AlphaFold mapping file, that included a function call to *retrieve_maxtemplate()*. The code below shows a snippet of the *utils.py* file with the definition of *retrieve_maxtemplate()*. The function checks if a date was provided and returns the correct time string. The returned string will be used as a placeholder value

```
def retrieve_maxtemplate(maxtemplate=""):
    if maxtemplate == "":
        return f""+str(datetime.date.today())
    else:
        return f""+maxtemplate
```

5.3 Drona Support Library

Drona Workflow Engine provides a support library with Python functions to assist in creating powerful and flexible environments, and can be called directly from any function in utils.py. The support library currently contains the following functions.

```
drona_add_message()
drona_add_additional_file()
drona_add_mapping()
```

These functions communicate directly with the Drona Engine. We will discuss these library functions in more detail in the following subsections.

The support library is defined in file *utils.py* in the Drona Engine system folder. Additional functions can be added if needed. Depending on how Drona Engine was installed, only people with elevated access, such as system administrators, might have write access to this file.

5.3.1 Adding Messages.

In Section 3.3, we discussed how the preview window can include immediate feedback using various types of messages (error messages, warnings, and notes). The support library provides the following function to add any of these messages.

```
drona_add_message(message, type)
```

Where **message** is a string containing the message and **type** is the type of the message, with valid values being "error", "warning", "note". For convenience and backward compatibility, the Drona support library also provides the following functions:

```
drona_add_error(message)
drona_add_warning(message)
drona_add_note(message)
```

5.3.2 Adding Template Files Dynamically.

The previous section discussed how the *additional_files.json* file is used to specify template files to be included in the workflow. Although very useful, there are scenarios where statically adding files is insufficient. For advanced workflows, it is not uncommon for input values to dictate which additional files should be included. The support library provides the following function to add template files to the workflow dynamically.

```
drona_add_additional_file(file, name, index)
```

where *file* is the path, either relative to the environment directory or absolute, *name* is the name shown in the preview tab, and *index* is the tab position (-1 indicates to exclude the file from the preview).

5.3.3 Adding Mappings Dynamically.

Typically, placeholders are defined statically in file *map.json*. However, to provide ultimate flexibility in adding mappings, the Drona support library offers a function to add a mapping dynamically.

```
drona_add_mapping(key,value)
```

where *key* is the placeholder name to add, and *value* is the string associated with the placeholder. It is worth noting that the *value* string can contain other placeholders as long as these placeholders are static.

5.3.4 Example Support Library Call.

The following Python code shows a snippet of a function in the AlphaFold *utils.py* file. The code is truncated for brevity and to make it fit. The Python function checks if the researcher picked AlphaFold3, and in that case, it adds the correct GPU job file to the workflow, using the *drona_add_additional_file* support library function. It also adds a mapping for RECYCLES, which is only used for AlphaFold3 using the *drona_add_mapping* support library function. Finally, the code snippet adds a warning message using the *drona_add_message* support library function if the researcher didn't provide a JSON input file

```
def process_alpahfold3(version,json,recycles):
    if version != "alphafold3":
        return ""

    drona_add_additional_file("alphafold3-gpu.job", \
                              "alphafold3-gpu")

    drona_add_mapping("RECYCLES",str(recycles))
    drona_add_mapping("runcommand",driver_alphafold3)

    if json == "":
        drona_add_message("No input JSON specified. \
                           Create one in preview  \
                           window", "warning")
    return ""
```

6 Managing Environments

In Section 3 we introduced the Drona GUI and showed the concept of environments to run scientific workflows. In Section 4 we discussed the structure and the various parts comprising an environment in great detail. In this section, we will briefly discuss how Drona manages environments and how researchers can share their Drona environments with their peers and the community.

6.1 Drona Environment Repository

In section 3.1 we showed how a researcher can import environments. Figure 2 showed the import window with available environments. The environments are stored in an external public GitHub repository [16], completely independent of the Drona Workflow Engine repository. Drona will pull in the environments from this external repository and show them in the import window. When a researcher clicks the "Add" button, Drona retrieves the requested environment from the repository and copies it into the researcher's directory.

6.2 Sharing Environments

Drona provides multiple options for sharing environments. Researchers can simply share their Drona environment directly with their peers, who can copy it into their local Drona environments directory. This is especially useful when the researcher has created or customized their own environment. To share a Drona environment with the community, the researcher can request that their Drona environment be included in the public Drona environments repository. The researcher will provide a metafile containing information about the Drona environment and create a pull request.

7 Scientific Workflow Monitoring

Monitoring and managing a workflow is as important as generating and executing that workflow, and it is common for the researcher to monitor its progress using batch-scheduler commands in a Linux shell. In a typical HPC setting, this includes checking job status, elapsed runtime, inspecting logs, and sometimes canceling a running job.

GUI tools, such as OOD and OpenComposer, provide some of this job-listing functionality in the GUI. Although useful, the information provided is primarily batch job-related and does not include any workflow-specific details. Additionally, these GUI job-listing tools struggle to handle workflows like AlphaFold, or scientific pipelines in general, that consist of multiple jobs, in an elegant manner.

We argue that the managing and monitoring phase is, or should be, an integral part of a scientific workflow. That is the philosophy Drona adheres to. A Drona environment contains a wealth of information about the scientific workflow it represents. This knowledge can be leveraged to provide detailed workflow-specific details in addition to batch-job-related information. For example, an AlphaFold3 environment can include form elements that show the progress of the prediction, display Pickle file information, print warning, and error messages, and show GPU utilization. The researcher who creates or adapts a Drona environment determines the relevant information for managing that specific workflow and includes the matching form elements, and if needed, driver logic to the environment. Drona provides the tools and building blocks to do this.

The screenshot in Figure 8 shows a simple example of how a Drona AlphaFold environment can include monitoring. In this case, as a proof of concept, we extended the AlphaFold environment to allow the user to monitor submitted workflows.

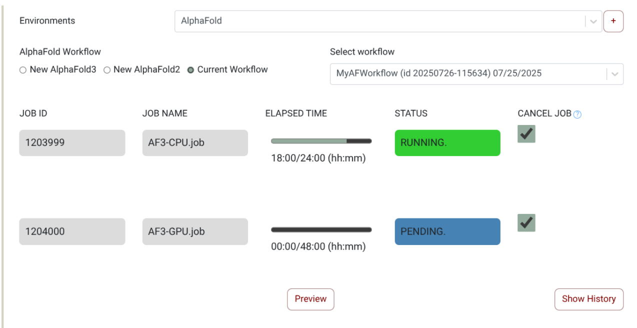

Figure 8: Screenshot of monitoring an AlphaFold3 job, showing information about both Slurm jobs.

The figure shows the researcher selected to monitor existing jobs and then chose a specific workflow. Once a specific workflow is selected, the Drona environment displays information for both CPU and GPU jobs, including job ID, name, elapsed time, state, and checkboxes that allow the researcher to cancel one or both jobs. In the screenshot, the researcher selected to cancel both jobs. In this case, the Drona environment will generate code for the driver to cancel the jobs (e.g., using scancel command in Slurm) once the researcher clicks the preview button.

8 Example Workflows

So far, we have only discussed how Drona can assist researchers in creating and generating scientific workflows. In this section, we will showcase two different types of workflows we have developed, demonstrating the versatility of Drona. These workflows build upon our previous work [9, 13].

8.1 IPU Tutorial

To showcase Drona's ability to simplify complex computational tasks, we created a Drona Tutorial environment for Graphcore Intelligent Processing Unit (IPU) training. During the tutorial, participants learn to port PyTorch [12] and TensorFlow [3](Keras [4]) models to the IPU through self-contained training modules that include all necessary materials, code files, and handouts. By eliminating the need for Linux shell familiarity, the Drona tutorial environment provides the following streamlined experience:

(1) The participant chooses the tutorial module from a dropdown menu. Once a module is selected, it will show the module goals and hints needed to complete the training module
(2) Drona generates the relevant files, including Python code snippets and bash scripts. The participant works on the exercises, switching between the preview window and hints if needed
(3) Participant clicks the submit button, and the code is executed directly on the IPU, the output is shown in the output pane.
(4) Participant continues with the next tutorial module.

.

This Drona training environment was used during the IPU tutorial held at the PEARC25 Conference in Columbus, Ohio [2], and during the ByteBoost Workshop in Pittsburgh, Pennsylvania [1].

8.2 AI Dashboard

To consolidate AI/ML capabilities and provide researchers with streamlined access to diverse computational workflows, we developed a unified AI Dashboard within Drona. This dashboard aggregates multiple expert-encoded environments, providing researchers with a single portal for common AI/ML tasks, rather than requiring navigation between separate tools. The dashboard integrates existing Drona environments for PyTorch workflows, Hugging Face [7] transformer models, and educational resources, providing the following cohesive experience:

(1) The researcher accesses the AI Dashboard and selects from available tasks such as "Train Custom Model," "Fine-tune Model from Hugging Face," or "IPU Tutorial."
(2) Upon task selection, Drona loads the appropriate environment and guides the researcher through the familiar Drona workflow interface.
(3) All generated files, results, and job history are organized within a centralized workspace, improving project management and reproducibility.
(4) The output from one tool can be used to inform another. For example, a model fine-tuned via the dashboard could immediately be made available for an inference task within the same environment.

9 Implementation

Drona is available in a public GitHub repository [18]. The front end is primarily built using React, and the back end is written using Flask. We provide Drona as a Phusion Passenger app within our OpenOnDemand (OOD) portal. The Drona repository can be cloned directly into OOD, either into the dev or sys environment. Setup will be done automatically through OOD; by default, the app will appear under the Jobs category.

Although we use OOD to host Drona, Drona does not depend on OOD to function. Alternative options for hosting include running as a local website, which we have done regularly for testing purposes, or as an app inside other dashboards and frameworks.

10 Future Work

We believe the Drona Workflow Engine core is stable and extremely versatile, allowing it to virtually generate any type of workflow, assisting both novice and experienced users with their research goals. Future work will mostly focus on improving usability and convenience.

Although Drona provides a rich set of form elements, there are potentially useful elements missing that we still want to add. The form generator component has already been utilized in other projects, and there will be opportunities to expand on it.

Similar to the building blocks (form elements) to create the GUI form, we plan to create a curated set of driver template functions that the researcher can choose from when creating a workflow. One simple example template would take a job name as input and submit the job to the Slurm scheduler. A more complex example template would take a list of job names as input and submit each subsequent job in the list to the Slurm scheduler, configured with a job dependency that prevents it from starting until the previously submitted job in the sequence has completed successfully, effectively creating a pipeline. These are just two examples. Other examples include templates for submitting to alternative schedulers, such as PBS, or to a Kubernetes cluster. Providing building blocks for drivers will further elevate the Drona design philosophy of creating workflows in a mostly declarative manner.

Drona is designed to be fully self-contained. That includes creating, updating, and sharing environments. Future work will include providing a **"Drona Env Management"** environment to manage, create, update, and share Drona environments created by researchers. Some preliminary work has already been done to create forms using drag-and-drop functionality.

One major focus area for future work is to add more Drona environments to the environments repository. Especially related to AI/ML and bio-related fields.

11 Summary

In this paper, we presented Drona Workflow Engine, both a tool designed to guide researchers in running their scientific workflows and a powerful framework for creating complex new workflows. Drona is researcher-focused, allowing researchers to have complete control over their workflows, including which workflows to import, create, or adapt, as well as share them with their peers. The intuitive and guided forms enable novice and experienced users to focus on their research. In addition, this results in a reduction in help requests for HPC centers.

Acknowledgments

We gratefully acknowledge the National Science Foundation for support of this work under award number 2112356 (ACES: Accelerated Computing for Emerging Sciences)

References

[1] 2025. ByteBoost Cybertraining Program Summer Workshop. Workshop held at the Pittsburgh Supercomputing Center, Pittsburgh, PA. https://www.stonybrook.edu/ookami/ByteBoost.php Accessed on September 19, 2025.
[2] 2025. PEARC '25: Workshops and Tutorials. https://pearc.acm.org/pearc25/workshops-and-tutorials/. Accessed on September 19, 2025.
[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.
[4] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras.
[5] Google Deepmind. 2020. AlphaFold homepage. https://deepmind.google/technologies/alphafold/.
[6] Dave Hudak, Doug Johnson, Alan Chalker, Jeremy Nicklas, Eric Franz, Trey Dockendorf, and Brian L. McMichael. 2018. Open OnDemand: A web-based client portal for HPC centers. *Journal of Open Source Software* 3, 25 (2018), 622. https://doi.org/10.21105/joss.00622
[7] Hugging Face Inc. 2025. Hugging Face. https://huggingface.co/ Accessed: August 6, 2025.
[8] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. 2021. Highly accurate protein structure prediction with AlphaFold. *Nature* 596, 7873 (2021), 583–589. https://doi.org/10.1038/s41586-021-03819-2
[9] Andrii Kryvenko, Marinus Pennings, and Honggao Liu. 2025. Democratizing Scientific Machine Learning through Expert-Encoded Workflow Templates. In *Proceedings of the Gateways 2025 Conference.* Accepted for publication.
[10] Masahiro Nakao, Riken. 2025. OpenComposer. https://github.com/RIKEN-RCCS/OpenComposer
[11] Abhinand Nasari, Hieu Le, Richard Lawrence, Zhenhua He, Xin Yang, Mario Krell, Alex Tsyplikhin, Mahidhar Tatineni, Tim Cockerill, Lisa Perez, Dhruva Chakravorty, and Honggao Liu. 2022. Benchmarking the Performance of Accelerators on National Cyberinfrastructure Resources for Artificial Intelligence / Machine Learning Workloads. In *Practice and Experience in Advanced Research Computing 2022: Revolutionary: Computing, Connections, You* (Boston, MA, USA) *(PEARC '22)*. Association for Computing Machinery, New York, NY, USA, Article 19, 9 pages. https://doi.org/10.1145/3491418.3530772
[12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
[13] Marinus Pennings, Honggao Liu, and Andrii Kryvenko. 2025. Generating Scientific Workflows With Drona Environments. In *Practice and Experience in Advanced Research Computing 2025: The Power of Collaboration (PEARC '25)*. Association for Computing Machinery, New York, NY, USA, Article 39, 5 pages. https://doi.org/10.1145/3708035.3736104
[14] Duy Pham, Kyle Hsu, Marinus Pennings, Tri Pham, and Phi Au. 2022. Extending Functionalities on a Web-based Portal for Research Computing. In *Practice and Experience in Advanced Research Computing 2022: Revolutionary: Computing, Connections, You* (Boston, MA, USA) *(PEARC '22)*. Association for Computing Machinery, New York, NY, USA, Article 37, 4 pages. https://doi.org/10.1145/3491418.3535182
[15] Joe Stubbs, Richard Cardone, Mike Packard, Anagha Jamthe, Smruti Padhy, Steve Terry, Julia Looney, Joseph Meiring, Steve Black, Maytal Dahan, Sean Cleveland, and Gwen Jacobs. 2021. Tapis: An API Platform for Reproducible, Distributed Computational Research. In *Advances in Information and Communication*, Kohei Arai (Ed.). Springer International Publishing, Cham, 878–900.
[16] Texas A&M High Performance Research Computing. 2025. Drona Environments GitHub repository. https://github.com/tamu-edu/dor-hprc-drona-environments
[17] Texas A&M High Performance Research Computing. 2025. Drona Form Elements Page. https://tamu-edu.github.io/dor-hprc-drona-composer/index.html
[18] Texas A&M High Performance Research Computing. 2025. Drona Workflow Engine GitHub repository. https://github.com/tamu-edu/dor-hprc-drona-composer