Pline: automatic generation of modern web interfaces for command-line programs

Andres Veidenberg[1] and Ari Löytynoja[1]

[1]Institute of Biotechnology, University of Helsinki, Finland

August 28, 2020

Abstract

Background: Bioinformatics software often lacks graphical user interfaces (GUIs), which can limit its adoption by non-technical members of the scientific community. Web interfaces are a common alternative for building cross-platform GUIs, but their potential is underutilized: web interfaces for command-line tools rarely take advantage of the level of interactivity expected of modern web applications and are rarely usable offline.

Results: Here we present Pline: a lightweight framework that uses program descriptions and web standards to generate dynamic GUIs for command-line programs. We introduce a plugin system for creating Pline interfaces and provide an online repository for sharing third-party plugins. We demonstrate Pline's versatility with example interfaces, a graphical pipeline for sequence analysis and integration to Wasabi web application.

Conclusions: Pline is cross-platform, open-source software that can be integrated to web pages or used as a standalone desktop application. Pline provides graphical interfaces that are easy to create and maintain, fostering user-friendly software in science. Documentation, demo website, example plugins and source code is freely available from `http://wasabiapp.org/pline`.

Keywords: Bioinformatics; Software Engineering; User Interfaces; Web Technologies

Background

Graphical user interfaces (GUI) are an essential part of modern software, providing users with an intuitive method to access all of the functionality offered by a program. A well-designed GUI guides users by, for example, displaying relevant actions, adapting to user input, providing info tooltips and other visual cues. Software developed for research purposes, however, rarely includes a GUI, relying solely on the command line interface (CLI). The CLI is appropriate for advanced users, who can then quickly integrate new software into existing pipelines, but it comes with a steep learning curve (that may never be overcome) for non-technical users. CLIs are sensitive to typing errors, requiring users to remember which command-line options need to be included to run a program effectively. Moreover, command-line tools generally have a single, well-defined function and therefore require additional programs to view or post-process the output, further complicating their usage. These issues may appear trivial, but, in practice, they dramatically limit the number of potential users to those who are already comfortable using the command-line.

GUIs make programs more user-friendly and ease the adoption of novel computational tools. While the inclusion of GUIs is in the interest of both developers and users, there is little incentive to code a native graphical user interface. Many journals are dismissive of the scientific contribution offered by more usable software and development teams in academia tend to be constrained in terms of members (Mangul et al., 2019). A common compromise is to set up an online service where a web interface is used to launch a CLI program on a remote server. This provides cross-platform, installation-free access to the software, but has many disadvantages: (i) it requires a web server, necessitating additional setup, programming and maintenance; (ii) it cannot be used offline; and (iii) it is limited by the fact that users need to share the available network and CPU resources. Furthermore, such web interfaces tend to be based on basic HTML forms with little interactivity to guide the users. While it is possible to develop more sophisticated web interfaces, this requires extensive knowledge of web technologies like CSS (Cascading Style Sheets) and JavaScript, and can take a long time to develop.

Although it is considerably easier to implement a web service than a native GUI, integrating multiple programs remains a challenge. Both the web interface and the server side code needs to be built for a specific CLI program. Moreover, most of the user input processing and related code typically resides on the server that is hidden from the users. This causes redundancy, because the web interfaces cannot be reused by third-party developers for modification and therefore need to be created from scratch. The issue could be addressed by standardizing communication between a graphical interface and the underlying CLI tool. Some specification standards have been developed, like the Common Workflow Language (cwl) or the Galaxy tools XML (Afgan et al., 2018), that define how to describe a CLI program in a text file. Scientific workflow management systems (e.g. Taverna (Wolstencroft et al., 2013) or CWL implementations like Arvados (arv)) utilize this information to integrate external programs and, in some cases (e.g. Galaxy), also for creating the graphical interface elements. However, these standards are optimized for building pipelines and therefore omit GUI-specific instructions. In addition, setting up and running a workflow management system together with its environment (e.g. a dedicated web server, system container or virtual machine) adds unnecessary overhead when used only for creating a GUI for a CLI program.

Here, we introduce Pline (" Plugin interface language"): a *specification* for describing command line programs and their interfaces, and a lightweight *framework* that uses these descriptions to generate interactive graphical user interfaces. By utilizing standardization and web technologies, Pline allows for creation of graphical user interfaces for command-line programs without programming, considerably lowering the bar to develop user-friendly software in science.

Pline aims to be a practical tool for adding GUIs for CLI programs. To that end, Pline-generated interfaces address many of the limitations in web-based GUI development by filling a number of requirements:

- *cross-platform*: Pline interfaces run on any device with a web browser
- *embeddable*: the interfaces can be placed into existing web content
- *programming-free*: interface source code is automatically generated
- *self-contained*: CLI programs and its GUI are can be run as a standalone application
- *user-friendly*: the graphical interfaces guide users with interactivity and info tooltips

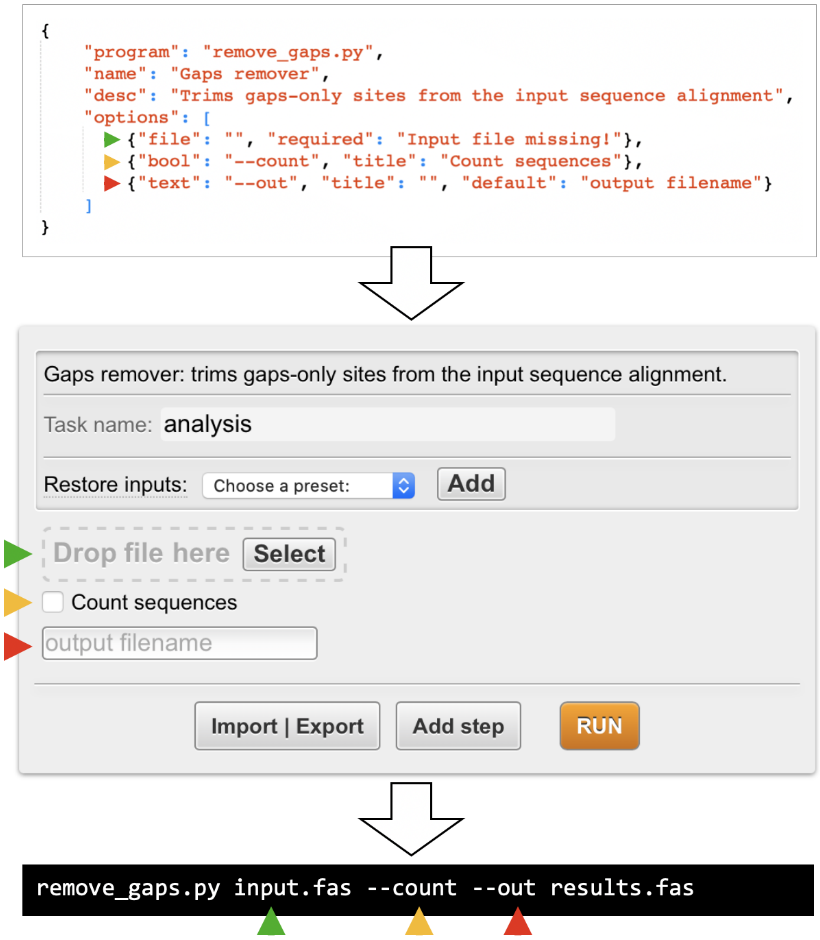
In this work, we utilize Pline in the context of bioinformatics, adding GUIs and a graphical pipeline to a number of command-line analysis tools. However, Pline's approach for generating GUIs is generic and can be applied to any field, including use cases outside the CLI domain.

Implementation

On technical level, Pline is a graphical user interface generator that wraps command-line programs to self-contained web applications. Each web application consists of three parts: Pline interface generator, a command-line executable together with its description file, and the Pline server module. The workflow of the web application can be divided to three steps:

1. Construction of a formal description of the underlying command-line program in a text file. This needs to be done only once for each program.
2. When the application is launched in a web browser, Pline reads the description file to generate its graphical user interface.
3. The GUI then incorporates the user input to construct the final CLI command and forwards it to the server module for execution.

This process is illustrated by a working example in Figure 1. Here, three input parameters for a command-line script is specified in the description file. Each parameter is then translated to the corresponding input element in the resulting web interface and placed in the specified order to the output command. Each of these steps is described in more detail in the following sections.



```
{
    "program": "remove_gaps.py",
    "name": "Gaps remover",
    "desc": "Trims gaps-only sites from the input sequence alignment",
    "options": [
      {"file": "", "required": "Input file missing!"},
      {"bool": "--count", "title": "Count sequences"},
      {"text": "--out", "title": "", "default": "output filename"}
    ]
}
```

Gaps remover: trims gaps-only sites from the input sequence alignment.

Task name: analysis

Restore inputs: Choose a preset:

□ Count sequences

output filename

```
remove_gaps.py input.fas --count --out results.fas
```

Figure 1: Example of a Pline application workflow. Pline uses a program description (top) to generate corresponding graphical interface (middle), which in turn produces the program launch command (bottom). Colored triangles highlight the location of the program parameters in each workflow stage.

Pline plugin API

For describing command-line interfaces, Pline defines an application programming interface (API) specification based on JSON standard. JSON (JavaScript Object Notation) is a common format for representing structured data in human-readable text (jso). A Pline web application can include one or more *Pline plugins* – CLI programs and their JSON descriptions stored in text files. These files inform Pline on how to launch the underlying program as well as how to draw its user interface.

With the plugin API, a CLI program is defined using a list of valid command-line arguments, where each argument is described with property/value pairs and grouped with nested brackets. Pline utilizes this notation to effectively describe interactive GUIs: the data (a set of properties with values) defines the underlying functionality, while its structure (the order and nesting of parameters) reflects the placement of resulting interface elements. Since most of the API properties are optional, basic interfaces are quick to construct. The only compulsory data fields are the executable name and the type (or name) of each input argument. A simple example is shown in Figure 1. Here, the JSON description specifies a python script that expects an input file (as a positional argument), followed by a boolean flag named "–count" and a text input called "–out" (both optional, named arguments). Using Pline API, this information is presented with a compact piece of JSON: `{"file": ""}`, `{bool: "--count"}`, `{"text": "--out"}`. The rest of the properties shown in the figure specify optional functionality. For example, `"required"` adds a check for filling the file input and the accompanying error message, while `"title"`, `"default"` and `"desc"` will display relevant information in specific places in the resulting interface.

Input arguments are often related. The set of valid arguments, their expected values, and the interpretation of (even the same) values by a command-line program may change depending on how the user has filled some other, related input argument. In the Pline JSON format, the network of linked inputs[1] is described by setting rules for the input properties that support it. For example, instead of a fixed default value for an input, a rule can derive the value from another input. These rules are written as conditionals – English-like if-else sentences (or alternatively, JavaScript statements), where the action of the rule is defined by the property that the rule is attached to. To illustrate, the static `"default"` property of the `--out` parameter in Figure 1 could be replaced with a dynamic one: `"default": "'foo.txt' if count, else 'bar.txt'"`. This rule would swap the default parameter value (and show it in the text input) depending on whether the checkbox element (controlling the `--count` parameter) is ticked. In addition to setting the default value, Pline supports conditionals for dynamically formatting or fixing an input value, for specifying an output file name and for enabling or disabling input elements and element groups. Together with other advanced features like input filters, error messages and merged values, the Pline API allows for describing even highly complex command-line interfaces. To customize the resulting GUI, the API also specifies properties for adding icons, labels, documentation, HTML markup and CSS rules.

[1] When used on its own, the term *input* refers to a CLI program argument and its representation in the JSON, the GUI, and the command-line form (see Figure 1).

Interface rendering

After a plugin JSON has been added to a Pline web application, it's ready to launch its GUI. For that, Pline includes an interface generator that implements the plugin API, translating program descriptions into graphical user interfaces at runtime. The generator is written in JavaScript, runs natively inside any modern web browser, and is incorporated to web pages as a library. The library exposes functions like `addPlugin()` for importing plugin descriptions and `plugin.draw()` for rendering interfaces for the imported plugins. The first function translates a plugin description (given as URL or raw text) to an internal data model, whereas the second one converts the model to the final interface (HTML and JavaScript code) and places it to a chosen container in the web page. Integrating Pline to any web content is therefore straightforward - the default web page in Pline web application is a blank container that populates its interface by calling these functions for any description files it finds from its plugins folder.

In the example in Figure 1, Pline has translated each parameter in the JSON to corresponding input element in the interface. Pline supports both simple input types like text, files, checkboxes or selection lists, and advanced ones that merge or modify values from linked inputs. In addition to input elements, Pline interface includes a header that displays the program description and provides an option to name the program execution session, as well as to save or restore sets of user input values.

A Pline-generated GUI is not static – the HTML interface is bound to an internal data model and event listeners that enforce the dependency rules between the program parameters and adjust the interface according to user interactions. User input is tracked in real-time: as soon as a tick-box is clicked or a number is typed, the interface updates accordingly, e.g. by hiding, revealing, or changing the values of all the linked input elements. The conditionals in the Pline JSON therefore provide a quick way to construct sophisticated interfaces that hide invalid inputs and guide the user through program configuration options. In addition to generating standalone GUIs, Pline can chain multiple interfaces together, forming a pipeline – a set of commands executed in succession. The information about input and output files in the program description is used to control the data flow between the pipeline steps. The current state of a single interface or a full pipeline can be stored to a file and distributed as a reusable Pline pipeline with pre-filled input values.

By default, Pline stacks interface elements (e.g. inputs and pipeline sections) into a single column. This layout is optimized for tight spaces, like windowing systems in web applications or mobile device screens. In plugin JSON, the inputs can be rearranged to rows and static or collapsible sections by grouping the elements with brackets. For further interface customization (e.g. element spacing, dimensions and the color scheme), the styling rules in the included CSS file can be modified, overriden by the host website or replaced altogether (e.g. with a CSS library like Bootstrap(boo)).

Command-line management

The “Run” button in the plugin interface initiates the third workflow step: the application checks for missing user input, prepares the input files and constructs the terminal command for launching the CLI program (or pipeline). For the interface shown in Figure 1, it would display an error message “Input file missing” next to the empty file input, since the associated parameter is marked as compulsory in the source JSON. Filling all the inputs would produce a command with three parameters as shown on the figure together with a confirmation message on the submission button.

Next, the CLI program is launched by the Pline-generated web application. Since the web browser security sandbox prevents direct command-line access, the program launch data is passed on to a backend server for execution. Pline includes a lightweight python script that acts as a server module to launch the commands, either on a local computer or over the web. The Pline server accepts the command data sent by the interface via an HTTP request, sanitizes the input, and manages the execution process. It also supports follow-up requests to send execution status updates back to the interface, pause, cancel or resume running pipelines, or send email notifications after a command or pipeline has finished.

Results

Plugin repository

New graphical interfaces are added to a Pline web application by supplying the corresponding JSON description files, either by copying them into the designated plugin directory (when using with standalone web application), or by passing JSON data directly to the Pline interface generator (using `addPlugin()`). In principle, a Pline GUI can be generated for any command-line executable, including installed programs that are available system-wide. However, the JSON description is written for a specific version of a program and it is recommended that the matching executable is distributed together with its description file, forming a plugin. For collecting and sharing Pline plugins, we have created a public repository as part of the Pline homepage (hom). At the time of writing, it contains 11 plugins for CLI programs used in phylogenetic and short-read sequence analyses. Plugins can be downloaded both as JSON files (useful for website integration or as a template for new plugins), as well as standalone applications (includes Pline, the JSON and the CLI program). The repository webpage is also an example of integrated Pline, which is used to generate a working interface for each plugin in the list. Similarly, a live version of the GUI shown in Figure 1 is available on the Pline front page. The plugin files are sourced from a dedicated GitHub repository, where third-party plugin contributions can be made via Git pull requests. Updates and additions to the plugins list are automatically shown on the Pline repository webpage.

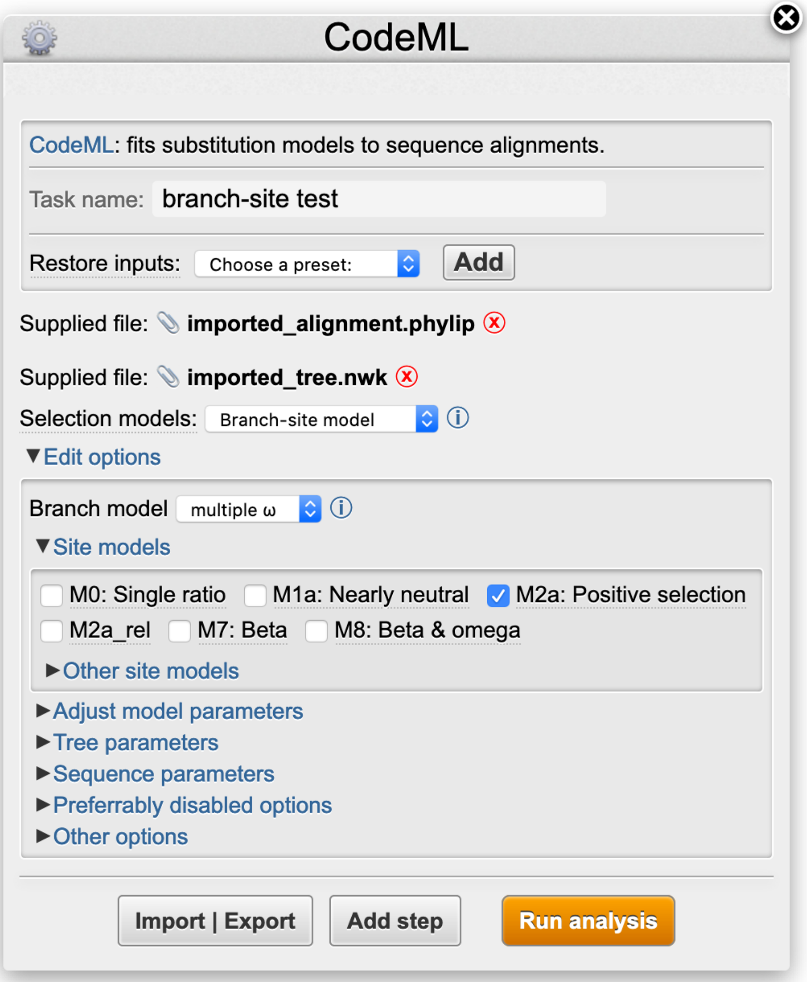
The downloadable Pline applications are designed to be installation-free and work across many different operating systems. The JSON program description files are platform-agnostic, the interface generator runs on any device with a modern web browser (including mobile devices), and the server module supports both Python 2.7 and 3 environments (which are preinstalled on most MacOS and Linux systems). However, binary command-line executables are compiled to run on a specific operating system, so a Pline plugin should to include an executable for each target system. To reduce file size, the plugins on the repository page are provided in multiple versions for different operating systems (currently for Linux and MacOS).

Integration into Wasabi

Pline does not contain binary executables, consisting of human-readable text files written in HTML, JavaScript and CSS (Python for the server module). Similar to other web pages, a Pline application is easy to modify and customize (e.g. changing the GUI appearance by editing CSS rules). In addition, Pline-generated interfaces are self-contained web elements and the JavaScript library can be extended with custom functions to modify any step in the interface generation process. This flexibility is especially useful for integrating Pline into existing web pages.

As an example of extensive integration, we added Pline to Wasabi, a web-based environment for evolutionary sequence analyses (was). With Pline, we were able to integrate external analysis programs into Wasabi's graphical interface as plugins without having to write program-specific interfaces and related code from scratch. Wasabi-specific features were added to the Pline interface generator as extensions. For example, an additional function in the plugin registration step makes a pop-up menu showing all the available plugins in Wasabi. Other additions automatically convert user-supplied files to the correct format, show the status updates of running programs in the Wasabi menu bar, and collects the resulting output files in an analysis database. The extensions are available as open-source software at Wasabi homepage (was).

Since Wasabi is designed for evolutionary sequence analysis, the list of integrated plugins include tools for related tasks: PRANK (Löytynoja and Goldman, 2005), PAGAN (Löytynoja et al., 2012) and MAFFT (Katoh and Standley, 2013) for multiple sequence alignment; FastTree (Price et al., 2010) for phylogenetic inference; CodeML (Yang, 2007) for tests of positive selection. All these plugins are also available in the plugins repository. Out of these programs, CodeML has the most complex interface and serves as a comprehensive example that utilizes a majority of the options available in the Pline API. The CodeML interface in Wasabi windowing system is shown in Figure 2. The CodeML plugin JSON (and therefore its interface) includes multiple presets – stored sets of pre-filled argument values – that are useful for running common configurations of selection models and related parameters. When users select a preset, the interface hides or reveals the relevant inputs, fills these with default values and enables the corresponding models from a set of tick boxes. When the user modifies an input that is part of the selected preset, the interface checks for dependencies and changes the preset selection as the combination of the input values no longer matches the initial preset. As an example of proxy inputs, the set of model selection tick boxes are converted to their corresponding command-line form as a single argument that consists of a string of numbers representing the selected model. As CodeML takes the input arguments through a configuration file, the `"configfile"` and `"valuesep"` properties in the JSON data instruct Pline to store the argument values as a whitespace-delimited text file and to launch the program with the file path as its only input argument.



CodeML

CodeML: fits substitution models to sequence alignments.

Task name: branch-site test

Restore inputs: Choose a preset:

Supplied file: imported_alignment.phylip

Supplied file: imported_tree.nwk

Selection models: Branch-site model

Branch model multiple ω

□ M0: Single ratio □ M1a: Nearly neutral ☑ M2a: Positive selection
□ M2a_rel □ M7: Beta □ M8: Beta & omega

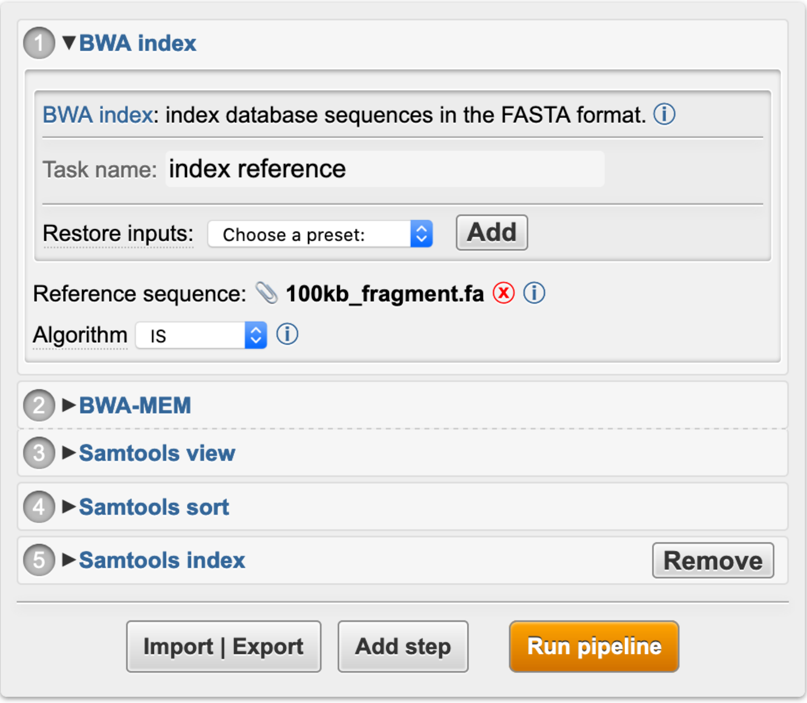
Figure 2: Pline interface for the CodeML plugin, rendered inside the Wasabi interface window (note the top title bar). The input files (marked with the paperclip icon) have been automatically supplied by the Wasabi environment. Info icons and underlined text show relevant tooltips on mouseover.

Example pipeline

The "Add a step" button at the bottom of the Pline interface allows users to quickly build a pipeline of commands by picking programs from the list of imported plugins. The plugin interfaces are drawn as a stack of collapsible sections, numbered by the order of execution. When all the inputs have been filled as needed, the resulting pipeline can be stored in a JSON file with the "Import/Export" button. The button also allows for restoring pipelines from existing JSON files to be rerun (e.g. with different input files), providing a convenient system for reusable and distributable graphical pipelines for command-line programs.

The Pline repository website includes an example analysis pipeline that maps short sequencing reads to a reference genome. It consists of four steps, starting with BWA-MEM (Li and Durbin, 2009) for mapping reads, followed by a series of Samtools commands (Li et al., 2009) for converting, sorting and indexing the sequencing reads pileup (see Figure 3). The downloadable plugin package includes all the files needed for standalone execution, including JSON descriptions of the pipeline and plugins, the Pline interface generator and example input sequence data. Double-clicking the Pline executable will launch the server module and open a web browser window with the graphical pipeline interface. Each of the pipeline sections can be expanded with a mouse click to examine and modify the pre-filled inputs as needed. For example, the merged sections for the first two steps indicate that these programs are launched as a piped command, where the output from the first program (*BWA-MEM*) is directly streamed as input for the second one (*Samtools view*). By changing the file input selection in the *Samtools view* interface from "pipe" to "standard output", the commands will be separated and the intermediate output file will be instead written to disk. After the included example files have been dragged to their respective file drop areas in the *BWA-MEM* interface, the pipeline can be run.

Although the web page container in the example pipeline package includes a minimal interface for displaying the status and the results of the pipeline after it has been launched, this functionality is outside the scope of the Pline interface generator. Instead, Pline offers a framework for web developers and scientists to integrate graphical interfaces for command-line programs to websites with very low effort, especially when the needed plugin descriptions are already available. However, the example code for the post-launch interface in the plugin package and Wasabi are open-source and can be used as-is or modified for custom integration. When Pline plugin is used as a standalone interface in desktop application form, the results retrieval interface is not needed as the files are directly accessible in the work directories specified by the Pline server configuration file.



BWA index: index database sequences in the FASTA format. ⓘ

Task name: index reference

Restore inputs: Choose a preset:

Reference sequence: 100kb_fragment.fa ⓧ ⓘ

Algorithm is ⓘ

Figure 3: Graphical interface of the example pipeline, generated after importing its JSON file. The first pipeline step (BWA index) has been expanded showing the file input that has been automatically filled with example data.

Discussion

The lack of graphical user interfaces is a common limitation of published computational tools, inhibiting their adoption by a wider scientific community. The importance of user-friendly analysis software in bioinformatics is indicated by the popularity of graphical environments for command-line analysis programs. For example, the commercially licensed Geneious software, which provides common command-line tools in a user-friendly analysis environment, advertises itself as the most cited software in molecular biology and sequence analysis (gen). Well-known open-source alternatives to Geneious include the UGENE (Okonechnikov et al., 2012) desktop application (which has a fixed set of tools) and web applications like qPortal (Mohr et al., 2018) or Galaxy (Afgan et al., 2018).

Since a significant amount of development and maintenance effort for GUI applications is dedicated to its interface (typically 45-50% as estimated by (Myers and Rosson, 1992)), it's understandable why most scientific software omit it. Pline provides a practical solution for the issue by translating simple JSON-based descriptions to fully-functional graphical GUIs. The concept of converting abstract data descriptions to user interfaces has been a subject of decades of research in the field of Model-Based User Interface Development (MBUID). In this approach, conceptual data models, which describe various aspects of an application (e.g. user tasks and UI presentation), are converted through multiple abstraction levels to final user interface (Meixner et al., 2011). Similar to Pline API, MBUID allows designers to describe user interfaces without worrying about the implementation details. However, in contrast to MBUID tools like EMUGEN (Brandl, 2002) and Mocadix (Vanderdonckt and Nguyen, 2019) that are designed to build heterogenous user interfaces for various use cases and environments, Pline has a much narrower scope. It trades the flexibility and complexity of MBUID for automation and ease of use, aiming to make GUI development for CLI programs as effortless as possible.

In essence, Pline is an interface generator that constructs form-based graphical interfaces for command-line programs. Over the years, many tools have been developed that overlap some aspects of Pline's functionality. Some examples like FormGen (Brandl and Klein, 1999) and Dynamic Forms (Girgensohn et al., 1995) are code generators for input forms, (pys) and Gooey (goo) aid python developers to add GUIs to their program, while UGUI (app) links web interfaces to CLI programs. Perhaps the most similar tool to Pline is Javamatic (Phanouriou and Abrams, 1997), which reads XML-based CLI descriptions to generate GUIs as Java applets. However, development of the tool was discontinued many years ago and it is no longer available for download.

Pline is a modern take on user-interface generators, building on the rapid development of web technologies and addressing some of the limitations in web-based GUI development. As a result, Pline interfaces are compatible with third-party web pages and also work as a standalone desktop application. In addition, Pline is well supported by its homepage (pli) that contains detailed documentation, tutorial videos and the plugins repository.

In addition, Pline's modular design allows for some of its functionality to be used independently of the CLI. The JSON plugin files, for example, could be used as information for generating human-readable documentation or converted to another format for use in workflow management systems. Also, the extensibility of Pline interface generator allows to use it for drawing form-based web interfaces outside of CLI domain. And since Pline-generated interfaces uses HTTP communication standard, the server module can easily be replaced with e.g. a backend from another website, further facilitating GUI integration. The Pline server module, however, is a lightweight and installation-free implementation of a web server that allows to run Pline interfaces as a standalone desktop application. With some modifications and using frameworks like Electron (ele), the Pline web application can be converted to a platform-specific native application with the accompanying user convenience and performance benefits.

Although making new Pline plugins does not require programming, manually writing the JSON data fields assumes some technical knowledge and, for larger plugins, can be a tedious task. This can be addressed in a couple of ways:

- Since the plugin JSON is a form of machine-readable program documentation, it could be converted to/from other similar formats (e.g. CWL (cwl)).
- Instead of writing a Pline plugin from scratch, it could be programmatically translated from e.g. a CWL description file, a CLI program help text or the Unix man page.
- To make the creation of Pline plugins as simple as possible, the JSON could be constructed using a graphical web page (that itself could be made with the help of Pline).

The converters and a graphical builder tool for making Pline plugins are the subject of future work.

Conclusions

Pline addresses the challenges in development of GUIs for command-line tools with a lightweight framework that utilizes simple data formats, code generation, and modern web technologies. This results in dynamic user interfaces that work cross-platform, including mobile devices. The JSON-based program descriptions allows creating, maintaining and sharing sophisticated interfaces without programming skills. We hope that the lower threshold of building graphical user interfaces earns Pline significant community support, resulting in a wide variety of graphical interfaces available in the online repository and promoting user-friendly software in science.

Availability and requirements

- **Project name:** Pline
- **Project home page:** http://wasabiapp.org/pline
- **Operating systems:** Platform independent
- **Programming language:** JavaScript, Python
- **Other requirements:** Web browser (Chrome, Safari, Firefox), Python 2.7+ or 3.0+
- **License:** MIT

List of abbreviations

- **API** - Application Programming Interface
- **CLI** - Command-line Interface
- **CSS** - Cascading Style Sheets
- **GUI** - Graphical User Interface
- **MBUID** - Model-Based User Interface Development

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Availability of data and materials

The datasets analysed in the example pipeline are available in the Pline repository (http://wasabiapp.org/pline/downloads).

Competing interests

The authors declare that they have no competing interests.

Funding

Not applicable.

Authors' contributions

AV designed and implemented Pline, example plugins and the example pipeline, and wrote the manuscript. AL supervised the project, designed the example pipeline and was a major contributor in writing the manuscript. All authors read and approved the final manuscript.

Acknowledgements

We thank Alan Medlar for his assistance in the writing process.

References

UGUI. https://ugui.io. Accessed: 01.08.2020.

Arvados. https://arvados.org. Accessed: 01.08.2020.

Bootstrap. https://getbootstrap.com. Accessed: 01.08.2020.

Common Workflow Language. https://commonwl.org. Accessed: 01.08.2020.

Electron. https://electronjs.org. Accessed: 01.08.2020.

Geneious. https://geneious.com. Accessed: 01.08.2020.

Gooey. https://github.com/chriskiehl/Gooey. Accessed: 01.08.2020.

Pline homepage. http://wasabiapp.org/pline. Accessed: 01.08.2020.

JSON. https://json.org. Accessed: 01.08.2020.

Pline homepage. http://wasabiapp.org/pline. Accessed: 2020-4-24.

PySimpleGUI. https://opensource.com/article/18/8/pysimplegui. Accessed: 01.08.2020.

Wasabi. http://wasabiapp.org. Accessed: 01.08.2020.

Enis Afgan, Dannon Baker, Bérénice Batut, Marius van den Beek, Dave Bouvier, Martin Cech, Chilton John, Dave Clements, Nate Coraor, Grüning Björn A, Aysam Guerler, Jennifer Hillman-Jackson, Saskia Hiltemann, Vahid Jalili, Helena Rasche, Nicola Soranzo, Jeremy Goecks, James Taylor, Anton Nekrutenko, and Daniel Blankenberg. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Res.*, 46(W1):W537–W544, jul 2018.

Alfons Brandl. Concepts for Generating Multi-User Interfaces Including Graphical Editors. In *Computer-Aided Design of User Interfaces III*, pages 167–178. Springer Netherlands, 2002. doi: 10.1007/978-94-010-0421-3_15. URL https://doi.org/10.1007%2F978-94-010-0421-3_15.

Alfons Brandl and Gerwin Klein. FormGen: A Generator for Adaptive Forms Based on EasyGUI. In *Human-Computer Interaction: Ergonomics and User Interfaces*, pages 1172–1176, 01 1999.

Andreas Girgensohn, Beatrix Zimmermann, Alison Lee, Bart Burns, and Michael E. Atwood. Dynamic Forms: An Enhanced Interaction Abstraction Based on Forms. In *IFIP Advances in Information and Communication Technology*, pages 362–367. Springer US, 1995. doi: 10.1007/978-1-5041-2896-4_60. URL https://doi.org/10.1007%2F978-1-5041-2896-4_60.

Kazutaka Katoh and Daron M Standley. MAFFT multiple sequence alignment software version 7: improvements in performance and usability. *Mol. Biol. Evol.*, 30(4):772–780, apr 2013.

Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler. *Bioinformatics*, 25(14):1754–1760, jul 2009.

Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Abecasis Goncalo, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. The Sequence Alignment/Map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, aug 2009.

Ari Löytynoja and Nick Goldman. An algorithm for progressive multiple alignment of sequences with insertions. *Proc. Natl. Acad. Sci. U. S. A.*, 102(30):10557–10562, jul 2005.

Ari Löytynoja, Albert J Vilella, and Nick Goldman. Accurate extension of multiple sequence alignments using a phylogeny-aware graph algorithm. *Bioinformatics*, 28(13):1684–1691, jul 2012.

Serghei Mangul, Lana S Martin, Eleazar Eskin, and Ran Blekhman. Improving the usability and archival stability of bioinformatics software. *Genome Biol.*, 20(1):47, feb 2019.

Gerrit Meixner, Fabio Paternò, and Jean Vanderdonckt. Past Present, and Future of Model-Based User Interface Development. *i-com*, 10(3):2–11, nov 2011. doi: 10.1524/icom.2011.0026. URL https://doi.org/10.1524%2Ficom.2011.0026.

Christopher Mohr, Andreas Friedrich, David Wojnar, Erhan Kenar, Aydin Can Polatkan, Marius Cosmin Codrea, Stefan Czemmel, Oliver Kohlbacher, and Sven Nahnsen. qPortal: A platform for data-driven biomedical research. *PLoS One*, 13(1):e0191603, jan 2018.

Brad A. Myers and Mary Beth Rosson. Survey on user interface programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92*. ACM Press, 1992. doi: 10.1145/142750.142789. URL https://doi.org/10.1145%2F142750.142789.

Konstantin Okonechnikov, Olga Golosova, Mikhail Fursov, and UGENE team. Unipro UGENE: a unified bioinformatics toolkit. *Bioinformatics*, 28(8):1166–1167, apr 2012.

Constantinos Phanouriou and Marc Abrams. Transforming command-line driven systems to Web applications, 1997.

Morgan N Price, Paramvir S Dehal, and Adam P Arkin. FastTree 2–approximately maximum-likelihood trees for large alignments. *PLoS One*, 5(3):e9490, mar 2010.

Jean Vanderdonckt and Thanh-Diane Nguyen. MoCaDiX. *Proceedings of the ACM on Human-Computer Interaction*, 3(EICS):1–40, jun 2019. doi: 10.1145/3331159. URL https://doi.org/10.1145%2F3331159.

Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid Belhajjame, Bacall Finn, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P Balcazar Vargas, Shoaib Sufi, and Carole Goble. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud, 2013.

Ziheng Yang. PAML 4: phylogenetic analysis by maximum likelihood. *Mol. Biol. Evol.*, 24(8):1586–1591, aug 2007.