The Journal of Open Source Software

omni-fig: Unleashing Project Configuration and Organization in Python

Felix Leeb [1¶]

1 Max Planck Institute for Intelligent Systems, Tübingen, Germany ¶ Corresponding author

DOI: 10.21105/joss.05350

Software

- Review
- Repository
- Archive

Editor: George K. Thiruvathukal

Reviewers:

- @julianpistorius
- @luciorq
- @jarrah42

Submitted: 25 January 2023
Published: 04 June 2024

Abstract

We present a lightweight package to take care of the configuration and organization of all your Python projects. Although `omni-fig` is well suited for projects of virtually any size or complexity, it is specifically designed for research projects where a small development team doesn't want to waste time on boilerplate code or a bespoke user interface. Nevertheless, the intuitive project structure encourages developers to good design practices making it easier for collaborators and prospective users to rapidly find and understand the core contributions they need. The feature-rich configuration system completely abstracts the implementation from the specification of parameters, so the developer can focus entirely on creating new functionality, while the user can quickly and precisely specify the desired functionality by composing modular config files, through the command line interface, or even in an interactive environment like Jupyter. `omni-fig` transforms the project organization and configuration from a bothersome distraction into a joy, thereby improving clarity while also accelerating development.

Introduction

One particularly promising trend in computational research, especially in machine learning, is that releasing the code associated with projects is becoming increasingly common (Pineau et al., 2021). Not only is this crucial for reproducibility (Guyon, n.d.), but this also fosters interdisciplinary research and progress in industry (Aho et al., 2020).

However, the needs of developers differ significantly from those of potential users. *Developers* (and especially researchers) prefer flexible, open-ended tools tailored for extensibility to prototype and synthesize novel methods. Meanwhile, *users* prefer tools that automate details and provide a simple interface that is digestible and directly applicable to some downstream problem. Even fellow researchers exploring new methods want to quickly understand the essence of the contribution rather than getting bogged down in some idiosyncrasies of a project. For example, in machine learning, this has given rise to a variety of development practices (Ebert et al., 2016; Treveil et al., 2020) and AutoML packages (He et al., 2021) to make cutting edge methods more highly accessible for a variety of real-world applications. However, the opaque rigid interfaces and product-oriented focus cater more to the end-user, thereby increasing friction in design and synthesis for developers.

Statement of Need

Project organization and configuration is an important albeit unglamorous part of every project. Neglecting this aspect can significantly impede prototyping and development as well as making the project less understandable and useable for any potential user or collaborator. Consequently, quite a few packages already exist to help with this task.

Perhaps the most similar to our proposed `omni-fig` is `hydra` (Yadan, 2019), which is a popular configuration framework that provides a feature-rich user interface, even including composing config files to some extent. However, from the developer's perspective, the project structure of `hydra` is more constrained and the configuration system is built on top of `OmegaConf` (*OmegaConf - Flexible Python Configuration System*, 2012), making more advanced features such as automatic instantiation of objects nonexistent or more cumbersome. Packages such as `OmegaConf` or `pydantic` (Colvin, 2019), on the other hand, focus more on integrations and performance but lack high-level features and a user-friendly interface. Slightly less comparable are a variety of packages designed for more specific applications with fixed project structures, such as `dynaconf` (Rocha, 2018), `gin-config` (Holtmann-Rice et al., 2018), and `confr` (Arro, 2022). Finally, there are some built-in libraries that are commonly used for the command-line interface and configuration, such as `argparse` (*Argparse - Parser for Command-Line Options, Arguments and Sub-Commands*, n.d.) and `configparser` (*Configparser - Configuration File Parser*, n.d.). However, these provide minimal features and scale poorly to more complex, evolving projects.

All too often, the trade-off between power and simplicity results in software (particularly research projects) containing cutting-edge features barely perceptible behind a limited or even nonexistent user interface. Here, a good project configuration framework can bridge the gap between developers and users. The user wants to select functionality in a concise and intuitive way while still allowing fine-grained control when needed. Meanwhile, the developer first and foremost wants the configuration system to seamlessly provide the necessary parameters without interfering with the broader design and implementation of the functionality.

Summary

Here `omni-fig` strikes a favorable balance in that it was designed from the ground up to cater specifically to the needs of both the *developer* and the *user*.

The primary way the *developer* interacts with `omni-fig` is by registering any functionality as a `script` (for functions), `component` (for classes), or `modifier` (for mixins), creating config files as needed, and then accessing config parameters using the config object at runtime. Once registered, the objects can be accessed anywhere in the project through the config object, thereby incentivizing the developer to register any functionality that can be customized by configuration. Meanwhile, since config files can easily be composed, the developer is incentivized to separate configuration in a modular way. Finally, at runtime, the developer doesn't have to worry about how the config parameters are specified (e.g., as a command line argument or in a config file), but can simply access them through the config object. This abstraction allows arbitrarily complex objects, even including mix-ins added dynamically at runtime (see `modifier`), to be instantiated automatically.

From the user's perspective, the modular config files and explicit registration of top-level functionality greatly improve the transparency of the project. For example, just running `fig -h` returns a custom help message displaying all registered scripts in the project. Then the high-level modularity of the config files allows the developer to effortlessly create demos by composing existing config files to showcase the key features of the project.

For more information, check out the documentation which includes an overview of the key features with examples. We are also continuously working on new features such as adding integrations and improving error messages. In any case, contributions and feedback are always very welcome!

Acknowledgments

This work was supported by the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039B, and by the Machine Learning Cluster of Excellence, EXC number 2064/1 – Project number 390727645. The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Felix Leeb. The authors would also like to thank Vincent Berenz for his feedback and suggestions, and Amanda Leeb for designing the omni-fig logo.

References

Aho, T., Sievi-Korte, O., Kilamo, T., Yaman, S., & Mikkonen, T. (2020). Demystifying data science projects: A look on the people and process of data science today. *International Conference on Product-Focused Software Process Improvement*, 153–167. https://doi.org/10.1007/978-3-030-64148-1_10

Argparse - parser for command-line options, arguments and sub-commands. (n.d.). https://docs.python.org/3/library/argparse.html

Arro, M. (2022). *Confr–a configuration system for machine learning projects*.

Colvin, S. (2019). *Pydantic - data parsing and validation using Python type hints*. Github. https://github.com/pydantic/pydantic

Configparser - configuration file parser. (n.d.). https://docs.python.org/3/library/configparser.html

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. *IEEE Software*, *33*(3), 94–100.

Guyon, I. (n.d.). *Artificial intelligence for all*.

He, X., Zhao, K., & Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, *212*, 106622. https://doi.org/10.1016/j.knosys.2020.106622

Holtmann-Rice, D., Guadarrama, S., & Silberman, N. (2018). *Gin config - provides a lightweight configuration framework for Python*. Github. https://github.com/google/gin-config

OmegaConf - flexible Python configuration system. (2012). Github. https://github.com/omry/omegaconf

Pineau, J., Vincent-Lamarre, P., Sinha, K., Larivière, V., Beygelzimer, A., d'Alché-Buc, F., Fox, E., & Larochelle, H. (2021). Improving reproducibility in machine learning research: A report from the NeurIPS 2019 reproducibility program. *Journal of Machine Learning Research*, *22*.

Rocha, B. (2018). *Dynaconf - configuration management for Python*. Github. https://github.com/dynaconf/dynaconf

Treveil, M., Omont, N., Stenac, C., Lefevre, K., Phan, D., Zentici, J., Lavoillotte, A., Miyazaki, M., & Heidmann, L. (2020). *Introducing MLOps*. O'Reilly Media.

Yadan, O. (2019). *Hydra - a framework for elegantly configuring complex applications*. Github. https://github.com/facebookresearch/hydra