The Journal of Open Source Software

anywidget: reusable widgets for interactive analysis and visualization in computational notebooks

Trevor Manz [1¶], **Nezar Abdennur** [2,3], **and Nils Gehlenborg** [1]

1 Department of Biomedical Informatics, Harvard Medical School, Boston, MA, USA **2** Department of Genomics and Computational Biology, UMass Chan Medical School, Worcester, MA, USA **3** Department of Systems Biology, UMass Chan Medical School, Worcester, MA, USA ¶ Corresponding author

DOI: 10.21105/joss.06939

Software

- Review
- Repository
- Archive

Editor: Hugo Ledoux

Reviewers:

- @kylebarron
- @ianhi

Submitted: 10 June 2024
Published: 23 October 2024

Summary

The **anywidget** project provides a specification and toolset for portable and reusable web-based widgets in interactive computing environments (Figure 1). First, it defines a standard for widget front-end code based on the web browser's native module system. Second, it provides tools to author, distribute, and execute these modules across web-based computing platforms. Since its release in February 2023, anywidget has steadily gained adoption. As of October 2024, nearly 100 new widgets have been created or ported to anywidget and published to the Python Package Index (PyPI), along with many standalone scripts and notebooks. These tools cover general-purpose visualization libraries (Heer & Moritz, 2024; Lekschas & Manz, 2024) as well as notebook integrations for applications in biology (Mark S. Keller et al., 2021; Manz et al., 2022, 2023; Manz, Lekschas, et al., 2024), mapping (Barron, 2024), astronomy (Boch & Desroziers, 2020), and education (Warmerdam, 2024). Anywidget has also been integrated into popular visualization libraries like Altair (VanderPlas et al., 2018), enhancing interactivity in notebooks and deepening user engagement with visualizations and code.

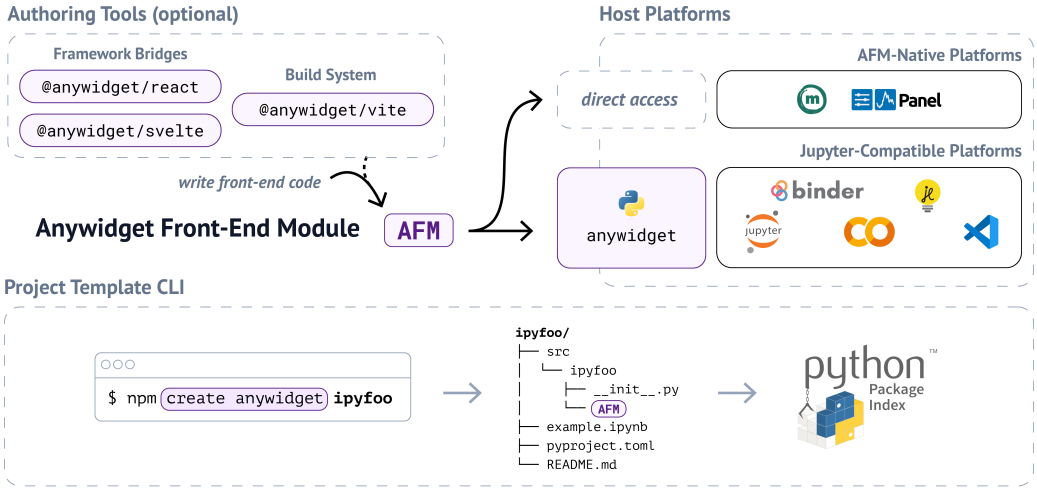

Figure 1: The anywidget project. Components highlighted in magenta. The Anywidget Front-End Module (AFM) is a specification for widget front-end code based on ECMAScript (ES) modules (Guo et al., 2023). AFM can be written in web-standard ES or with *authoring tools* that support popular front-end frameworks. The `anywidget` Python package adapts Jupyter-compatible platforms (JCPs) into AFM-compatible *host platforms*, enabling Jupyter Widgets to be authored and distributed with AFM. Other *host platforms* support AFM directly. The *project CLI* can be used to bootstrap new anywidget projects that are ready to publish to PyPI.

Statement of need

Computational notebooks are the preferred environment for interactive computing and data analysis. These platforms provide an interface where users can write, execute, and interact with code and data in real-time. Notebooks connect an interactive front end (typically a web browser) to an external system that runs the code, combining prose, executable snippets, and media. The widespread adoption of notebooks has driven the development of tools for integrating interactive visualizations within these environments (Z. J. Wang et al., 2024). The Jupyter project (Granger & Pérez, 2021; Kluyver et al., 2016; Perez & Granger, 2007) has become synonymous with computational notebooks, largely due to its widespread success in fostering an extensive ecosystem of community tools (e.g., for producing books (Holdgraf & others, 2022), presentation slides (F. Wang et al., 2023), and dashboards (e.g., Voilà)).

Approaches for authoring interactive visualizations across notebook platforms, including Jupyter, vary widely (Z. J. Wang et al., 2024). This inconsistency has led to fragmented methods for creating and distributing custom visualizations and interactive components, hindering the development of a composable solutions. The Jupyter Widgets system shows potential for aligning interactive visualization systems with the broader notebook ecosystem. However, the complexity and error-prone nature of authoring custom widgets has limited their adoption by the visualization community. Widget development is hindered by fragmented distribution and cumbersome development experience due to complex integration requirements from diverse environments (Manz, Gehlenborg, et al., 2024). Moreover, Jupyter Widgets are Jupyter-specific, with the widest support limited to Python kernels, leaving gaps in addressing new and alternative interactive computing environments. A universal protocol is needed to simplify authorship and support an ecosystem of pluggable interactive widgets.

Overview

A standard for widget front-end modules

The **Anywidget Front-End Module (AFM)** is a specification for widget front-end code based on ECMAScript (ES) modules, the built-in module system for web browsers.

```
export default {
  initialize({ model }) {
    // Add instance-specific event listeners
    return () => {
      // Clean up event listeners
    }
  },
  render({ model, el }) {
    // Render the widget
    return () => {
      // Clean up event listeners
    }
  },
};
```

Widget behavior is defined through methods for managing a widget's lifecycle, such as initialization, rendering, and destruction. Running AFM requires a *host platform* to load the module and call each of these lifecycle methods with a set of required interfaces. These interfaces include minimal APIs for the AFM to communicate with the host and modify the output user interface (UI).

Importantly, AFM does not impose specific implementations for widget state or UI management. By making host requirements explicit, it decouples widget front-end code from host

implementations, thereby improving widget portability (Manz, Gehlenborg, et al., 2024). With AFM, developers can author a widget by writing a web-standard ES module, either inline or in a separate file, without a build process (Figure 2a, top). For better ergonomics when creating UIs, developers can introduce a build step targeting AFM to utilize advanced tools (Figure 2a, bottom).

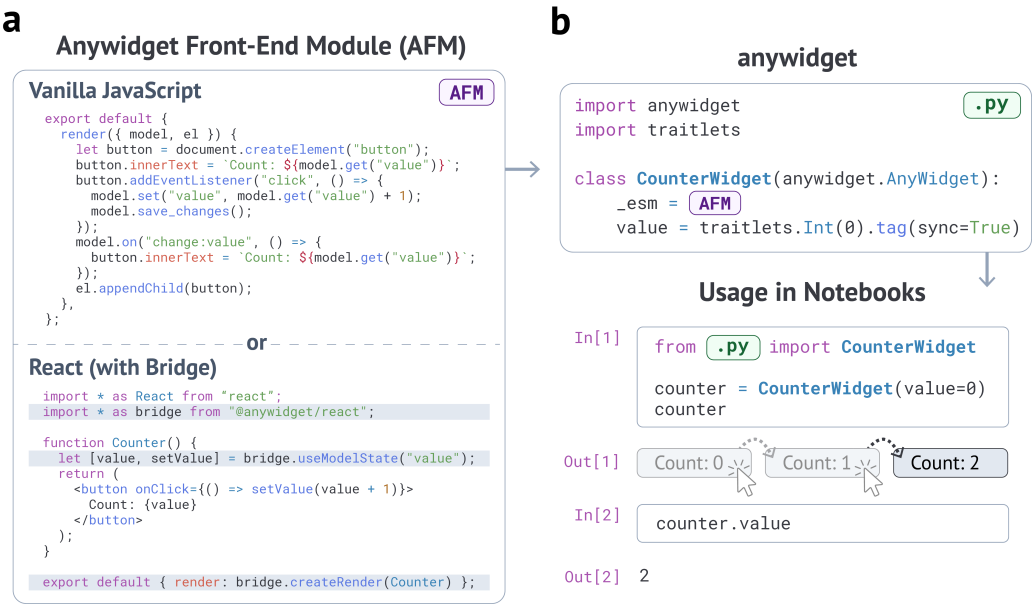

Figure 2: Authoring a custom Jupyter Widget with anywidget. (a) AFM can be authored in web-standard ECMAScript (top) or with a front-end framework using a bridge (bottom). (b) The anywidget Python package allows authoring custom Jupyter Widgets with AFM (top), usable across various JCPs (bottom).

While AFM is understood by the browser directly, much of web development today uses front-end frameworks, such as React or Svelte, which introduce non-standard syntax and unique paradigms for UI and state management. Rather than incorporate such frameworks into the AFM specification, the anywidget project provides several *framework bridges* to make it easier to author AFMs using frameworks (Figure 2a, bottom). These libraries provide utilities to use idiomatic APIs and constructs to manage widget state and to wrap those constructs into the AFM lifecycle methods used by host platforms. For example, anywidget's React bridge exposes a React-based declarative hook `useModelState` for accessing widget state and a function to convert a React component into an AFM export.

There are several advantages to supporting frameworks via bridges. First, AFM is more stable and minimal because it is not tied to a third-party library or framework. Second, it gives framework communities the opportunity to build integrations, distributing the maintenance burden and benefiting from framework-specific expertise. This plugin-based approach has seen success in projects like the local front-end development server Vite and the Python Flask web framework. Third, bridges can be updated and versioned independently from the AFM specification, meaning that changes cannot break host compatibility.

Supporting tools and ecosystem

Custom Jupyter Widgets

The main library for the project is `anywidget`, a Python package that simplifies the authoring and distribution of custom Jupyter Widgets using AFM (Figure 2b, top). Jupyter Widgets (Grout et al., 2024) are the official framework from Jupyter to extend notebooks with interactive

views and controls for objects that reside in the kernel. Since widgets are integral to Jupyter's architecture, they enjoy broad support across Jupyter-compatible platforms (JCPs) such as JupyterLab, Google Colab, and Visual Studio Code. However, developing and distributing cross-JCP widgets is complex and error-prone (Manz, Gehlenborg, et al., 2024). Anywidget addresses this complexity by providing the glue code to turn each JCP into an AFM-compatible host. This compatibility layer aligns AFM with Jupyter Widgets and the Python ecosystem, making anywidget a powerful tool for creating and distributing interactive widgets across platforms. Anywidgets can be remixed and reused with other custom Jupyter Widgets in notebooks, standalone HTML pages, and dashboarding frameworks (Figure 2b, bottom).

Tooling for authorship and distribution

To make widget development more enjoyable and accessible, the anywidget project offers additional development tools for widget authors. It allows for creating widgets directly within notebooks, enabling them to start as prototypes and evolve into full packages. Aligning with modern front-end tools, anywidget also implements hot module replacement (HMR) for live code editing development. HMR dynamically updates widgets without reloading the page or losing state, improving the developer experience and enabling rapid prototyping. To enable HMR, developers can set an environment variable in the notebook cell:

```
%%env
ANYWIDGET_HMR=1
```

Finally, the anywidget project includes a command line interface (CLI) for bootstrapping new anywidget projects that are ready to publish to PyPI (Figure 1, bottom). The CLI includes options for selecting front-end framework adapters and additional tools like TypeScript.

```
npm create anywidget@latest
```

Beyond Jupyter

AFM extends the widget ecosystem beyond Jupyter. Many popular web frameworks and dashboarding libraries support embedding Jupyter Widgets into their layout systems and interacting with their components. This support also extends to anywidgets, thanks to the Jupyter Widgets compatibility layer. AFM also provides opportunities for frameworks and platforms to add more specialized support, making better use of their respective internal state management and reactivity systems. For instance, marimo (Agrawal & Scolnick, 2024), a new reactive notebook for Python, has adopted AFM as the standard for its third-party plugin API. Similarly, the Panel web framework (Rudiger et al., 2024) supports using AFM to define custom components.

Efforts are underway to support AFM with other compute backends besides Python. For example, anyhtmlwidget (Mark S. Keller, 2024) brings anywidget concepts to R, enabling reusable AFM-based widgets for R documents and Shiny applications with bi-directional R-JavaScript communication. Additionally, the anywidget project implements AFM-based displays for the Deno kernel, a JavaScript and TypeScript runtime.

Availability

The anywidget project is released under an open-source MIT license, with all source code publicly available on GitHub (https://github.com/manzt/anywidget). The core Python library, anywidget, is packaged and distributed via the PyPI and conda-forge. The front-end adapter libraries, development tooling, and project-template CLI are distributed through the npm registry. The Deno Jupyter kernel integration is published to the JavaScript Registry (JSR). Further documentation about the project can be found at https://anywidget.dev.

Related work

Interactive notebook visualization tools vary widely in features and compatibility (Z. J. Wang et al., 2024). Some tools offer rich features (e.g., bi-directional communication) but rely on platform-specific APIs, limiting compatibility (Drosos et al., 2020; Jain et al., 2022; Li et al., 2023; F. Wang et al., 2023; Zhao et al., 2022). For example, frameworks like Bokeh and Streamlit enable custom extensions, but these integrations are tied to their specific frameworks and cannot be reused elsewhere. More simple approaches provide broader compatibility but lack features which meaningfully enrich user workflows. For example, using static templates or the NOVA framework (Z. J. Wang et al., 2022) offers wide compatibility, as the resulting HTML displays can be embedded in nearly any web-based notebook platform. However, this approach supports only client-side applications with one-way communication, meaning that only the initial visualization state can come from the notebook, without further updates from other cells. Other approaches, like ImJoy (Ouyang et al., 2019), offer a more unified architecture for building interactive visualizations with rich features across multiple platforms. However, it is an entirely separate computing platform with limited JCP integrations, not a framework for building reusable, modular visualization components.

Acknowledgements

We thank Talley Lambert for his technical contributions to the anywidget Python codebase and recognize Jan-Hendrik Müller for his significant community contributions and advocacy of the project. Our appreciation extends to the entire anywidget community and the Abdennur and HIDIVE labs for their helpful discussions.

Funding

TM, NA, and NG acknowledge funding from the National Institutes of Health (UM1 HG011536, OT2 OD033758, R33 CA263666, R01 HG011773).

References

Agrawal, A., & Scolnick, M. (2024). *Marimo*. https://github.com/marimo-team/marimo

Barron, K. (2024). *Lonboard*. https://github.com/developmentseed/lonboard

Boch, T., & Desroziers, J. (2020). ipyaladin: Enabling Aladin Lite in Jupyter notebooks. In P. Ballester, J. Ibsen, M. Solar, & K. Shortridge (Eds.), *Astronomical data analysis software and systems XXVII* (Vol. 522, p. 117).

Drosos, I., Barik, T., Guo, P. J., DeLine, R., & Gulwani, S. (2020). Wrex: A unified programming-by-example interaction for synthesizing readable code for data scientists. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–12. https://doi.org/10.1145/3313831.3376442

Granger, B. E., & Pérez, F. (2021). Jupyter: Thinking and storytelling with code and data. *Comput. Sci. Eng.*, *23*(2), 7–14. https://doi.org/10.22541/au.161298309.98344404/v3

Grout, J., Frederic, J., Corlay, S., & al., et. (2024). *ipywidgets: Interactive widgets for the Jupyter notebook*. https://github.com/jupyter-widgets/ipywidgets

Guo, S., Ficarra, M., Gibbons, K., & community, E. (2023). *ECMAScript® 2023 Language Specification* (14th ed.). https://262.ecma-international.org/14.0/

Heer, J., & Moritz, D. (2024). Mosaic: An architecture for scalable & interoperable data views. *IEEE Trans. Vis. Comput. Graph.*, *30*(1), 436–446. https://doi.org/10.1109/TVCG.2023.

3327189

Holdgraf, C., & others. (2022). *Jupyter Book and MyST: A community-led, extensible, modular ecosystem for creating computational narratives*. Zenodo. https://doi.org/10.5281/zenodo.7287626

Jain, N., Vaidyanath, S., Iyer, A., Natarajan, N., Parthasarathy, S., Rajamani, S., & Sharma, R. (2022). Jigsaw: Large language models meet program synthesis. *Proceedings of the 44th International Conference on Software Engineering*, 1219–1231. https://doi.org/10.1145/3510003.3510203

Keller, Mark S. (2024). *Anyhtmlwidget*. https://github.com/keller-mark/anyhtmlwidget

Keller, Mark S., Gold, I., McCallum, C., Manz, T., Kharchenko, P. V., & Gehlenborg, N. (2021). Vitessce: A framework for integrative visualization of multi-modal and spatially-resolved single-cell data. In *OSF Preprints*. https://doi.org/10.31219/osf.io/y8thv

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., & Willing, C. (2016). Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Scmidt (Eds.), *Positioning and power in academic publishing: Players, agents and agendas* (pp. 87–90). IOS Press. https://doi.org/10.3233/978-1-61499-649-1-87

Lekschas, F., & Manz, T. (2024). Jupyter Scatter: Interactive exploration of large-scale datasets. *Journal of Open Source Software*, *9*(101), 7059. https://doi.org/10.21105/joss.07059

Li, H., Ying, L., Zhang, H., Wu, Y., Qu, H., & Wang, Y. (2023). Notable: On-the-fly assistant for data storytelling in computational notebooks. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 1–16. https://doi.org/10.1145/3544548.3580965

Manz, T., Gehlenborg, N., & Abdennur, N. (2024). *Any notebook served: Authoring and sharing reusable interactive widgets*. OSF Preprints. https://doi.org/10.31219/osf.io/pyn7u

Manz, T., Gold, I., Patterson, N. H., McCallum, C., Keller, M. S., Herr, B. W., 2nd, Börner, K., Spraggins, J. M., & Gehlenborg, N. (2022). Viv: Multiscale visualization of high-resolution multiplexed bioimaging data on the web. *Nat. Methods*, *19*(5), 515–516. https://doi.org/10.1038/s41592-022-01482-7

Manz, T., L'Yi, S., & Gehlenborg, N. (2023). Gos: A declarative library for interactive genomics visualization in python. *Bioinformatics*, *39*(1). https://doi.org/10.1093/bioinformatics/btad050

Manz, T., Lekschas, F., Greene, E., Finak, G., & Gehlenborg, N. (2024). *A general framework for comparing embedding visualizations across class-label hierarchies*. https://doi.org/10.31219/osf.io/puxnf

Ouyang, W., Mueller, F., Hjelmare, M., Lundberg, E., & Zimmer, C. (2019). ImJoy: An open-source computational platform for the deep learning era. *Nat. Methods*, *16*(12), 1199–1200. https://doi.org/10.1038/s41592-019-0627-0

Perez, F., & Granger, B. E. (2007). IPython: A system for interactive scientific computing. *Comput. Sci. Eng.*, *9*(3), 21–29. https://doi.org/10.1109/MCSE.2007.53

Rudiger, P., Madsen, M. S., Hansen, S. H., Liquet, M., Andrew, Artusi, X., Bednar, J. A., B, C., Stevens, J.-L., Signell, J., Roumis, D., Deil, C., Paprocki, M., Wu, J., Mease, J., Arne, thuydotm, Amanieu, H.-Y., Coderambling, … TBym. (2024). *Holoviz/panel: Version 1.4.4* (Version v1.4.4). Zenodo. https://doi.org/10.5281/zenodo.11403810

VanderPlas, J., Granger, B., Heer, J., Moritz, D., Wongsuphasawat, K., Satyanarayan, A., Lees, E., Timofeev, I., Welsh, B., & Sievert, S. (2018). Altair: Interactive statistical visualizations for Python. *J. Open Source Softw.*, *3*(32), 1057. https://doi.org/10.21105/joss.01057

Wang, F., Liu, X., Liu, O., Neshati, A., Ma, T., Zhu, M., & Zhao, J. (2023). Slide4N:

Creating presentation slides from computational notebooks with human-AI collaboration. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 1–18. https://doi.org/10.1145/3544548.3580753

Wang, Z. J., Munechika, D., Lee, S., & Chau, D. H. (2024). SuperNOVA: Design strategies and opportunities for interactive visualization in computational notebooks. *Extended Abstracts of the 2024 CHI Conference on Human Factors in Computing Systems*, 1–17. https://doi.org/10.1145/3613905.3650848

Wang, Z. J., Munechika, D., Lee, S., & Chau, D. H. (2022). *NOVA: A practical method for creating notebook-ready visual analytics*. https://doi.org/10.48550/arXiv.2205.03963

Warmerdam, V. D. (2024). *Drawdata*. https://github.com/koaning/drawdata

Zhao, Z., Koulouzis, S., Bianchi, R., Farshidi, S., Shi, Z., Xin, R., Wang, Y., Li, N., Shi, Y., Timmermans, J., & Kissling, W. D. (2022). Notebook-as-a-VRE (NaaVRE): From private notebooks to a collaborative cloud virtual research environment. *Softw. Pract. Exp.*, *52*(9), 1947–1966. https://doi.org/10.1002/spe.3098