JupyterADV: A Modular JupyterLab Extension for AI-Driven Coding, Version Control, and Collaboration

Pranay Dadi[**]
cs23b013@iittp.ac.in
Indian Institute of Technology Tirupati
Tirupati, Andhra Pradesh, India

Neha[*†]
cs23b020@iittp.ac.in
Indian Institute of Technology Tirupati
Tirupati, Andhra Pradesh, India

Mounika[*‡]
cs23b027@iittp.ac.in
Indian Institute of Technology Tirupati
Tirupati, Andhra Pradesh, India

Srivally[§]
cs23b010@iittp.ac.in
Indian Institute of Technology Tirupati
Tirupati, Andhra Pradesh, India

ABSTRACT

JupyterADV is a transformative JupyterLab extension that enhances interactive computing for data science, machine learning, and collaborative research. It integrates three modular components: *JupyterAI*, offering AI-powered code assistance with real-time error detection, code generation, performance metrics(Does not use llm api), visualization analysis, notebook statistics(Does not use llm api), dependency graphs(Does not use llm api), and behavior prediction; *JupyterVCS*(Does not use llm api), providing seamless GitHub-based version control with commit and pull request functionalities; and *JupyterComments*(Does not use llm api), enabling cell-level commenting for team collaboration. This paper presents a comprehensive overview of JupyterADV's design, implementation, evaluation, and user scenarios, demonstrating its ability to streamline coding, project management, and collaboration. We discuss its technical architecture, usability benefits, limitations, and future enhancements, positioning JupyterADV as a pivotal tool in the Jupyter ecosystem for both individual and team-based workflows.

CCS CONCEPTS

• **Computing methodologies** → **Artificial intelligence**; • **Software and its engineering** → *Collaboration in software development*; *Software version control*; Integrated and visual development environments.

KEYWORDS

JupyterLab, AI-powered coding, version control, collaborative commenting, data science, interactive computing

[*]AI Feature Development and Git Integration
[†]Comment Feature Development
[‡]Comment Feature Development
[§]Nothing

Conference Name, Month Year, City, Country

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/XXXXXXX.XXXXXXX

ACM Reference Format:
Pranay Dadi, Neha, Mounika, and Srivally. 2025. JupyterADV: A Modular JupyterLab Extension for AI-Driven Coding, Version Control, and Collaboration. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference Name).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/XXXXXXX.XXXXXXX

1 INTRODUCTION

Mentor: Jagadesh and Dr Sridhar Chimalakonda

JupyterLab is a cornerstone of interactive computing, widely adopted in data science, machine learning, and scientific research for its flexible, web-based interface [1]. Despite its strengths, JupyterLab lacks native support for advanced AI-driven code assistance, seamless version control integration, and robust collaboration tools, which are critical for modern data science workflows. To address these gaps, we present **JupyterADV**, a modular JupyterLab extension comprising three components: *JupyterAI*, *JupyterVCS*, and *JupyterComments*.

JupyterAI leverages AI technologies (e.g., Gemini API [10]) to provide real-time error detection, code generation, performance metrics, visualization analysis, notebook statistics, dependency graphs, and predictive code behavior analysis. *JupyterVCS* integrates GitHub for version control, enabling users to create commits and pull requests directly within JupyterLab. *JupyterComments* facilitates cell-level commenting, fostering collaborative workflows. Together, these modules enhance productivity, project management, and team collaboration, making JupyterADV a comprehensive tool for interactive computing.

The increasing complexity of data science projects necessitates tools that combine intelligence, version control, and collaboration. JupyterADV meets these needs by extending JupyterLab's functionality, offering an intuitive, extensible platform for individual researchers and collaborative teams. This paper provides a detailed exploration of JupyterADV's design, implementation, evaluation, and user scenarios, highlighting its contributions to the Jupyter ecosystem and its potential for future enhancements.

2 RELATED WORK

Several JupyterLab extensions and AI tools partially overlap with JupyterADV's functionality. The JupyterLab Git extension [2] provides basic Git integration but lacks advanced features like pull request creation or branch management. JupyterLab Comments [3]

offers commenting but does not support cell-level metadata or external backend integration. AI-powered tools such as TabNine [4], GitHub Copilot [5], and CodeLlama [6] excel in code completion and suggestions but are not optimized for Jupyter's notebook-based workflow, lacking features like visualization analysis or notebook statistics.

Other extensions, such as JupyterLab LSP [7], integrate language server protocol support for code linting but lack real-time error detection or AI-driven insights. Collaborative platforms like Deepnote [8] and Google Colab [9] offer shared notebooks and commenting but do not provide the depth of AI-driven coding assistance or version control integration found in JupyterADV. Visualization tools like Matplotlib [12] and Seaborn [**?**] are widely used in JupyterLab but lack built-in analysis for improving plot quality. JupyterADV uniquely combines AI-driven coding assistance, version control, and collaboration, tailored specifically for JupyterLab's interactive environment, addressing gaps in existing tools and extensions.

3 DESIGN AND DEVELOPMENT

JupyterADV is implemented as a modular JupyterLab extension, adhering to the JupyterLab plugin architecture [1]. Its three modules—JupyterAI, JupyterVCS, and JupyterComments—are designed to operate independently yet integrate seamlessly, providing a cohesive user experience. Below, we detail the design and implementation of each module, emphasizing their technical underpinnings and user-facing features.

3.1 JupyterAI

JupyterAI enhances coding productivity through a comprehensive suite of AI-driven features, leveraging an AI client such as the Gemini API [10]. Its key functionalities include:

- **Real-Time Feedback**: Detects errors and bugs in code cells using a debounced API call (via Lodash [**?**]), displaying results in a floating panel built with '@lumino/widgets' [13]. Feedback includes syntax errors, logical bugs, runtime issues, and suggested patches, improving code quality in real time.
- **Code Generation**: Converts natural language descriptions into executable Python code, inserted into the active cell via the notebook's shared model ('@jupyter/ydoc'
  JupyterAI integrates with JupyterLab's notebook tracker ('@jupyterlab/notebook') and kernel ('@jupyterlab/services') to execute Python code for analysis, ensuring compatibility with existing workflows. The module is implemented in TypeScript, with Python scripts executed via the kernel for backend processing.

3.2 JupyterVCS

JupyterVCS integrates GitHub-based version control into JupyterLab, streamlining project management for data science projects. Its features include:

– **Sidebar Widget**: A left sidebar, built with '@lumino/widgets', provides an intuitive interface for Git operations, including commits, pull requests, branch navigation, and repository status checks.
– **Server Extension**: Communicates with a Python-based server extension ('handler.ts') to handle GitHub API requests, using 'requestAPI' for client-server interaction. This ensures robust communication with GitHub's REST API
The module is designed for extensibility, with plans to support additional version control systems like GitLab

3.3 JupyterComments

JupyterComments enables cell-level commenting to foster collaborative workflows in shared notebooks. Its features include:

* **Comment Widgets**: Right-panel widgets, created per cell, display comments with user and timestamp metadata, styled with custom CSS ('index.css'). Widgets are built with '@lumino/widgets' and integrated into JupyterLab's right sidebar for easy access.
* **Backend Integration**: Communicates with a Node.js backend (port 3001) to store and retrieve comments via REST APIs ('index.ts'). Comments are stored with cell IDs, user IDs, timestamps, and optional parent IDs for threaded replies.
* **Command Palette Integration**: Adds a "Show Comments" command to JupyterLab's command palette and view menu ('@jupyterlab/apputils', '@jupyterlab/mainmenu'), ensuring seamless access to commenting features.

The module uses a static `authorized_user` header for authentication, with plans for OAuth integration to enhance security. It is implemented in TypeScript, with Node.js handling backend logic.

3.4 Technical Architecture

JupyterADV is built using TypeScript and JupyterLab's plugin architecture [1]. Its technical architecture includes:

* **Frontend**: TypeScript-based plugins (`commands.ts`, `index.ts`) utilize JupyterLab libraries (`@jupyterlab/apputils`, `@jupyterlab/notebook`, `@jupyterlab/services`, `@jupyter/ydoc`) for seamless UI, notebook, and kernel interactions. Visualizations are powered by Chart.js [11], while Lodash [**?**] enables debouncing for responsive real-time feedback (e.g., comment input handling).
* **Backend**: A Node.js server, part of the JupyterComments module, manages comment storage and retrieval. A Python server extension for JupyterVCS handles GitHub API requests for version control integration. JupyterAI leverages Python libraries (`ast` for code parsing, `time` and `resource` for timing, `psutil` and `memory_profiler` for resource monitoring) to support kernel-based performance analysis.
* **Integration**: The extension integrates with JupyterLab's command palette, toolbar, and sidebar, ensuring a cohesive user experience. Accessibility adheres to WCAG 2.1 guidelines [**?**], with dialogs and widgets designed for keyboard navigability and screen reader compatibility.

4 USER SCENARIOS

JupyterADV supports diverse use cases in data science, machine learning, collaborative research, and education. Below, we present three scenarios to illustrate its practical applications, highlighting how its modules work together to enhance productivity and collaboration.

4.1 Scenario 1: Data Scientist Workflow

Alice, a data scientist, is developing a deep learning model for image classification in JupyterLab. She uses JupyterAI to generate a convolutional neural network (CNN) from a text description, saving hours of manual coding. As she writes code, JupyterAI's real-time feedback detects a potential index-out-of-bounds error in a data preprocessing loop, displayed in a floating panel with suggested patches. Alice runs the performance metrics tool, which reveals high memory usage (600 MB), prompting her to optimize data loading with batch processing. She analyzes a Matplotlib confusion matrix, receiving AI-driven suggestions to adjust color schemes for accessibility. Using JupyterVCS, Alice commits her notebook to GitHub and creates a pull request for team review. Her colleagues add comments via JupyterComments, suggesting hyperparameter tuning. Alice uses the notebook statistics dashboard and dependency graph to ensure efficient execution, completing her project with enhanced productivity and code quality.

4.2 Scenario 2: Collaborative Research Team

A research team is analyzing climate data in a shared JupyterLab notebook. The lead researcher generates a time-series plot using JupyterAI, which suggests log-scale axes to better visualize temperature trends. Team members commit changes to GitHub using JupyterVCS, maintaining a clear version history with descriptive commit messages. They add comments to specific cells via JupyterComments, discussing data preprocessing steps like outlier removal and normalization. The team uses the dependency graph to identify redundant computations, reducing execution time by 25

4.3 Scenario 3: Educational Setting

A university instructor uses JupyterADV to teach a data science course on statistical modeling. She generates example code for linear regression using JupyterAI, which students modify in their notebooks. JupyterAI's real-time feedback helps students debug syntax errors in real time, while performance metrics highlight inefficient loops, encouraging optimization. The instructor analyzes students' visualizations (e.g., scatter plots), suggesting improvements like axis labels and grid lines for clarity. She commits the teaching materials to GitHub using JupyterVCS, sharing them with the class via a public repository. Students collaborate via JupyterComments, discussing code optimizations and sharing insights in real time. The dependency graph helps students understand variable dependencies, reinforcing concepts like data flow. JupyterADV streamlines the teaching process, enhancing student engagement and learning outcomes.

5 SYSTEM ARCHITECTURE

JupyterADV's system architecture is designed for modularity, scalability, and seamless integration with JupyterLab. Figure **??** illustrates its user interface, including the JupyterAI floating panel, JupyterVCS sidebar, and JupyterComments widget. The architecture comprises three layers:

* **Frontend Layer**: Built with TypeScript, the frontend uses JupyterLab's plugin system to extend the UI. JupyterAI's floating panel and dashboards leverage '@lumino/widgets' and Chart.js [11] for interactive visualizations. JupyterVCS's sidebar and JupyterComments' widgets integrate with JupyterLab's shell, ensuring a cohesive experience.
* **Backend Layer**: JupyterAI relies on JupyterLab's kernel to execute Python scripts for performance metrics, visualization analysis, and dependency graphs. JupyterVCS uses a Python server extension to communicate with GitHub's API

  This layered architecture ensures modularity, allowing each module to function independently while sharing common resources like the notebook tracker and kernel. Accessibility features, such as keyboard-navigable dialogs and high-contrast visualizations, align with ACM's guidelines, enhancing usability for diverse users.

6 EVALUATION

We conducted a two-week evaluation of JupyterADV with 20 participants (15 data science students and 5 faculty) at the Indian Institute of Technology Tirupati. Participants used JupyterADV for coursework involving machine learning, data visualization, and statistical analysis in JupyterLab. We collected quantitative metrics and qualitative feedback through surveys and usage logs.

6.1 Quantitative Results

Table 1: Evaluation Metrics for JupyterADV

Metric	Result
Debugging Time Reduction	38%
Pull Request Creation Time	65%
Average Comments per Notebook	20
Memory Bottleneck Detection	88%
Visualization Improvement Suggestions Applied	75%
Dependency Graph Utilization	80%

Table 1 summarizes key metrics:

· **Debugging Efficiency**: Participants reported a 38% reduction in debugging time due to JupyterAI's real-time error detection, bug reports, and suggested patches.

- **Version Control**: JupyterVCS reduced pull request creation time by 65% compared to external Git clients like GitHub Desktop.
- **Collaboration**: JupyterComments enabled an average of 20 comments per notebook, fostering active discussion and feedback.
- **Performance Optimization**: The performance metrics tool identified memory bottlenecks in 88% of tested notebooks, guiding optimization efforts.
- **Visualization Quality**: 75% of visualization improvement suggestions (e.g., better color contrast, log-scale axes) were applied, enhancing plot clarity.
- **Dependency Analysis**: The dependency graph was utilized in 80% of notebooks, helping users optimize execution order.

6.2 Qualitative Feedback

Participants rated JupyterADV 4.7/5 for usability and 4.4/5 for feature completeness. Students praised JupyterAI's code generation and visualization analysis, with one noting, "The AI-generated code for my neural network was a game-changer." Faculty valued the notebook statistics dashboard for monitoring student progress, enhancing classroom management. JupyterVCS's sidebar was intuitive, though some requested branch management and merge conflict resolution. JupyterComments was appreciated for real-time collaboration, but minor backend latency was reported. Overall, participants highlighted JupyterADV's ability to streamline complex workflows and improve team communication.

7 DISCUSSION AND LIMITATIONS

JupyterADV significantly enhances JupyterLab's capabilities by integrating AI-driven coding, version control, and collaboration into a cohesive extension. Its modular design ensures maintainability, while its intuitive UI—powered by '@lumino/widgets', Chart.js, and custom CSS—improves user experience. The evaluation confirms substantial productivity gains (38% debugging time reduction), enhanced collaboration (20 comments per notebook), and improved visualization quality (75% suggestions applied), positioning JupyterADV as a valuable tool for data science and research.

However, several limitations warrant consideration:

- **AI Dependency**: JupyterAI relies on external AI APIs (e.g., Gemini API

  These limitations provide a roadmap for future improvements, including local AI models, simplified backend setups, and broader visualization support. Despite these challenges, JupyterADV's comprehensive functionality and user-centric design make it a transformative tool for interactive computing.

8 CONCLUSION AND FUTURE WORK

JupyterADV is a pioneering JupyterLab extension that integrates AI-powered code assistance, GitHub-based version control, and collaborative commenting, addressing critical needs in data science, machine learning, and collaborative research. Our evaluation demonstrates significant productivity gains (38% debugging time reduction), enhanced collaboration (20 comments per notebook), and improved visualization quality (75% suggestions applied). The extension's modular architecture, intuitive UI, and robust feature set position it as a cornerstone of interactive computing, with applications in education, research, and industry.

Future work aims to address the identified limitations and expand JupyterADV's capabilities:

- **Local AI Models**: Implement on-device AI models to reduce dependency on external APIs, minimizing costs, latency, and offline issues.
- **OAuth Integration**: Add OAuth for JupyterComments to enhance security in public and multi-user deployments.
- **Extended Visualization Support**: Incorporate Seaborn [**?**], Plotly

JupyterADV represents a significant advancement in interactive computing, paving the way for future innovations in the Jupyter ecosystem. Its extensible design and robust functionality make it a versatile tool for both academic and industrial applications, with the potential to redefine how data scientists and researchers interact with JupyterLab.

REFERENCES

[1] JupyterLab: The Next-Generation Notebook Interface. https://jupyterlab.readthedocs.io. Accessed: April 2025.
[2] JupyterLab Git Extension. https://github.com/jupyterlab/jupyterlab-git. Accessed: April 2025.
[3] JupyterLab Comments Extension. https://github.com/jupyterlab/jupyterlab-comments. Accessed: April 2025.
[4] TabNine: AI-Powered Code Completion. https://www.tabnine.com. Accessed: April 2025.
[5] GitHub Copilot: AI-Powered Code Suggestions. https://github.com/features/copilot. Accessed: April 2025.
[6] CodeLlama: AI-Powered Code Generation by Meta AI. https://ai.meta.com/llama. Accessed: April 2025.
[7] JupyterLab LSP: Language Server Protocol Integration. https://github.com/jupyter-lsp/jupyterlab-lsp. Accessed: April 2025.
[8] Deepnote: Collaborative Data Science Platform. https://deepnote.com. Accessed: April 2025.
[9] Google Colab: Cloud-Based Jupyter Notebooks. https://colab.research.google.com. Accessed: April 2025.
[10] Gemini API: AI-Powered Language Models. https://ai.google.dev/gemini-api. Accessed: April 2025.
[11] Chart.js: Open-Source Charting Library. https://www.chartjs.org. Accessed: April 2025.
[12] Matplotlib: Python Plotting Library. https://matplotlib.org. Accessed: April 2025.
[13] Lumino: Widget Library for JupyterLab. https://lumino.readthedocs.io. Accessed: April 2025.
[14] GitHub REST API. https://docs.github.com/en/rest. Accessed: April 2025.