PROCEEDINGS OF SCIENCE

Agent-based code generation for the Gammapy framework

Dmitriy Kostunin,[a,*] **Vladimir Sotnikov,**[b] **Sergo Golovachev,**[c] **Abhay Mehta,**[a] **Tim Lukas Holch**[a] **and Elisa Jones**[a]

[a] *Deutsches Elektronen-Synchrotron DESY, 15738 Zeuthen, Germany*
[b] *JetBrains Limited, 8046 Paphos, Cyprus*
[c] *JetBrains GmbH, 80639 München, Germany*

E-mail: dmitriy.kostunin@desy.de

The proceedings text was generated with a large language model (LLM), using the agent's source code as a reference.

Software code generation using Large Language Models (LLMs) is one of the most successful applications of modern artificial intelligence. Foundational models are very effective for popular frameworks that benefit from documentation, examples, and strong community support. In contrast, specialized scientific libraries often lack these resources and may expose unstable APIs under active development, making it difficult for models trained on limited or outdated data. We address these issues for the Gammapy library by developing an agent capable of writing, executing, and validating code in a controlled environment. We present a minimal web demo and an accompanying benchmarking suite. This contribution summarizes the design, reports our current status, and outlines next steps.

39th International Cosmic Ray Conference (ICRC2025)
15–24 July 2025, Geneva, Switzerland

ICRC 2025
The Astroparticle Physics Conference
Geneva July 15-24, 2025

*Speaker

1. Introduction

Large Language Models (LLMs) [1] are increasingly capable of writing non-trivial scientific software, but turning free-form assistance into reproducible *analysis scripts* still requires domain knowledge and guard-rails [2, 3][1][2]. In the domain of gamma-ray astronomy this gap is amplified by: (i) domain-specific data formats (Data Level 3, DL3), (ii) fast-moving APIs, and (iii) the need to run code in a controlled environment with the correct data mounted [4].

Gammapy [5] is an open-source Python package for high-energy gamma-ray astronomy built on the scientific Python stack. It provides a uniform platform to reduce and model DL3 data from different instruments, with background-estimation methods and Poisson maximum-likelihood fitting, and is used across the community, including next-generation Cherenkov Telescope Array Observatory (CTAO) tooling.

We present an **agent for Gammapy** that generates, executes, and self-repairs analysis scripts until they run successfully. The agent follows three principles:

- **Strong contracts in prompting.** The system message encodes rules such as "return one complete Python script", "import all dependencies", "do not call plotting display functions", and "do not select observations via `TARGET_NAME`". Rules are enforced at generation time and by validation.
- **Tight integration with the analysis stack.** The agent exposes the data location via an environment variable and executes code in a sandboxed workspace; stdout/stderr are captured, and timeouts protect against runaway jobs.
- **Iterative verification.** Generated code is executed; failures are summarized and fed back to the model. The loop stops when validation succeeds or a configured attempt budget is exhausted.

The implementation is a small, testable Python package (`gammapygpt`) with a benchmarking suite and a minimal web user interface (UI). It targets everyday Gammapy tasks (listing observations, spectral extraction with reflected regions, 3D binned analyses, quick-look maps), prioritizing clarity over cleverness. The text below reflects the not yet public code accompanying this paper.

2. Agent design for Gammapy analysis

Figure 1 sketches the workflow. A user prompt is expanded into a governed chat history; the model returns a single Python script; the script is executed in a controlled environment; the result is validated and, if needed, a concise error summary triggers the next iteration.

2.1 System architecture

The repository is organized in small modules:

[1] https://github.com/luo-junyu/Awesome-Agent-Papers
[2] https://agenticscience.github.io/

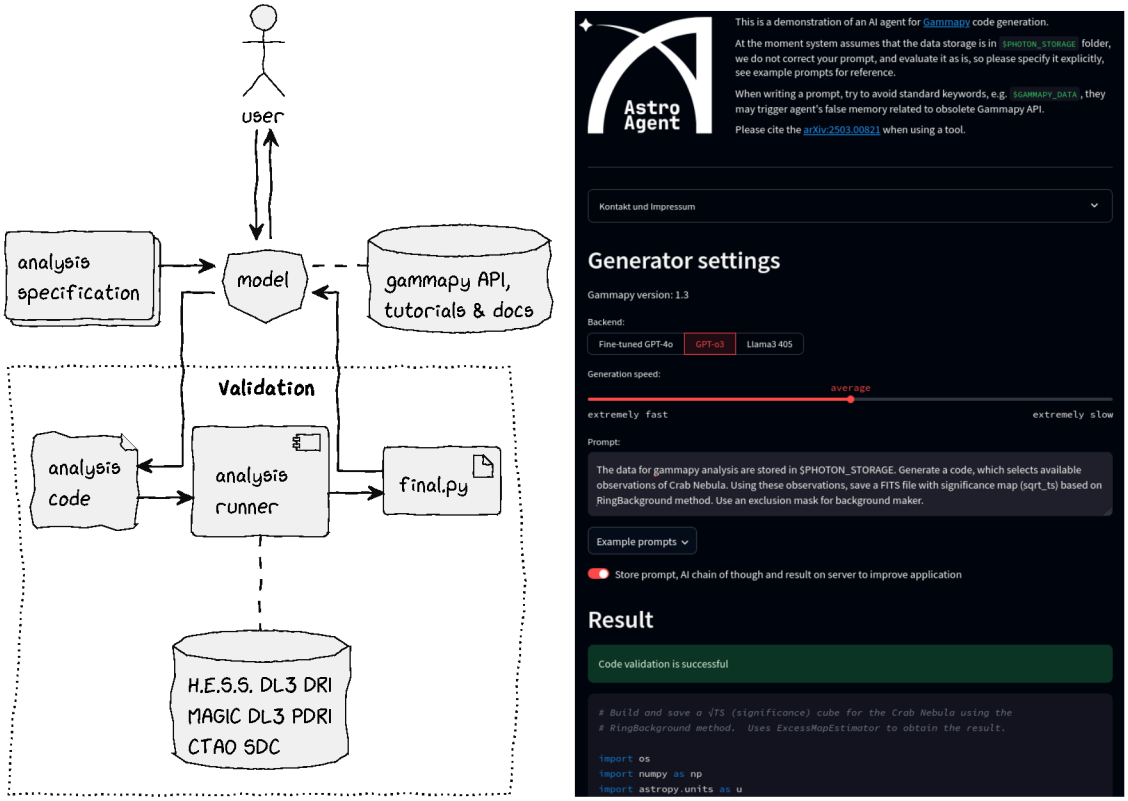

Figure 1: *Left:* Block diagram of the agent. Solid arrows form the generation–execution–validation loop; dashed arrows indicate retrieval of contextual snippets (tutorials, examples). *Right:* Screenshot of the Streamlit prototype (`https://majestix-vm8.zeuthen.desy.de`).

Configuration. A `Config` model stores *backends* (model name, base URL, API key, embedding model, and the preferred reasoning effort), timeouts, and filesystem locations. The data location is published via the environment variable `PHOTON_STORAGE`; users can override paths at runtime. A simple CLI saves/loads configuration files and provides helpers to fetch public H.E.S.S. DL3 data [6].

Messages and prompting. The system message encodes non-negotiable rules:

- return *exactly one* Python code block (no prose),
- import all requirements; avoid interactive plotting,
- avoid selecting observations via `TARGET_NAME`,
- prefer current Gammapy idioms.

For compatibility with tutorials and examples the system text still mentions `GAMMAPY_DATA`, while the agent itself publishes `PHOTON_STORAGE`. Both are accepted by the validator.

Runner and tool use. The *Runner* constructs the conversation (system + user + optional RAG context), calls the model with the configured reasoning effort, and enforces a lightweight "*return_python*" tool so the output is a single script without backticks or prose. The script is executed

and validated. On failure, the Runner appends a new user turn containing a compact traceback tail and hints; iteration continues until success or the attempt budget is reached.

Code execution. Scripts are written to a temporary file and executed under a pared-down environment that includes the data pointer (`PHOTON_STORAGE`). Stdout/stderr are captured, and a hard time limit aborts runaway jobs. When a `prefix` directory is configured, each attempt persists the generated script and the chat transcript for auditability.

Retrieval-augmented generation. An optional RAG layer indexes selected Gammapy tutorials into an in-memory Qdrant[3] collection using OpenAI embeddings. Top-k snippets (subject to a score threshold) are injected as additional user context. The index can be rebuilt deterministically from the same sources.

Utilities and data handling. Notebook-oriented tutorial scripts are pre-processed (plots stripped; IPython magics removed) to respect the "no plotting" contract and ensure headless execution. All data access in generated code is expected to resolve via `PHOTON_STORAGE`.

2.2 Interfaces

Two thin interfaces sit on top of the same core:

- a **CLI** with sub-commands to generate code, download datasets, and (optionally) build the RAG index;
- a minimal **Streamlit**[4] web app where users can choose a backend, set the generation attempt budget via a "speed" slider, toggle message persistence, and run prompts. The app displays the latest script and execution log and stores the conversation in a run folder.

2.3 Validation and safety

Validation is pluggable. By default an execution is *valid* if the process returns code 0; additional domain checks are supported (e.g. verifying expected numerical outputs). All runs are *offline*: the executor has no network access and only sees the mounted data path. Timeouts and filesystem prefixes are configured centrally, making runs reproducible.

2.4 Reproducibility

Each attempt yields a folder containing: the prompt, the message log, the generated script, raw stdout/stderr, and the validation outcome. This turns successful generations into reusable examples and keeps failures easy to diagnose.

3. Benchmarks

Benchmarking is essential to quantify model performance for `Gammapy` code generation. We include a harness in `gammapygpt.benchmark` that executes generated scripts in isolated environments, records iteration traces, and applies numerical validators with explicit tolerances where

[3] https://qdrant.tech/
[4] https://streamlit.io/

needed. We compare two OpenAI reasoning models, `o3` and `gpt-5` [7], both at the highest available reasoning effort.

Tasks and criteria. Each task consists of (i) producing runnable code and (ii) meeting a task-specific check. Examples include:

- *ObservationList*: select observations for a given source and print the number of observations (exact integer match).
- *ReflectedSignificance*: compute a reflected-region significance for a source.
- *ReflectedSpectrum*: perform a reflected-region spectral extraction and report the total energy flux and spectral index (floats with tolerance).
- *Source3DAnalysis*: reduce all available observations of a source to a `MapDataset` and fit a spatial–spectral model (considered a pass if the script runs end-to-end under the timeout, due to complexity).

Harness details. Every iteration logs exception classes, a compact traceback tail, tokens (input, cached input, output, and reasoning tokens), and the attempt index at which validation passed. RAG is optionally enabled, injecting top-k tutorial snippets when available.

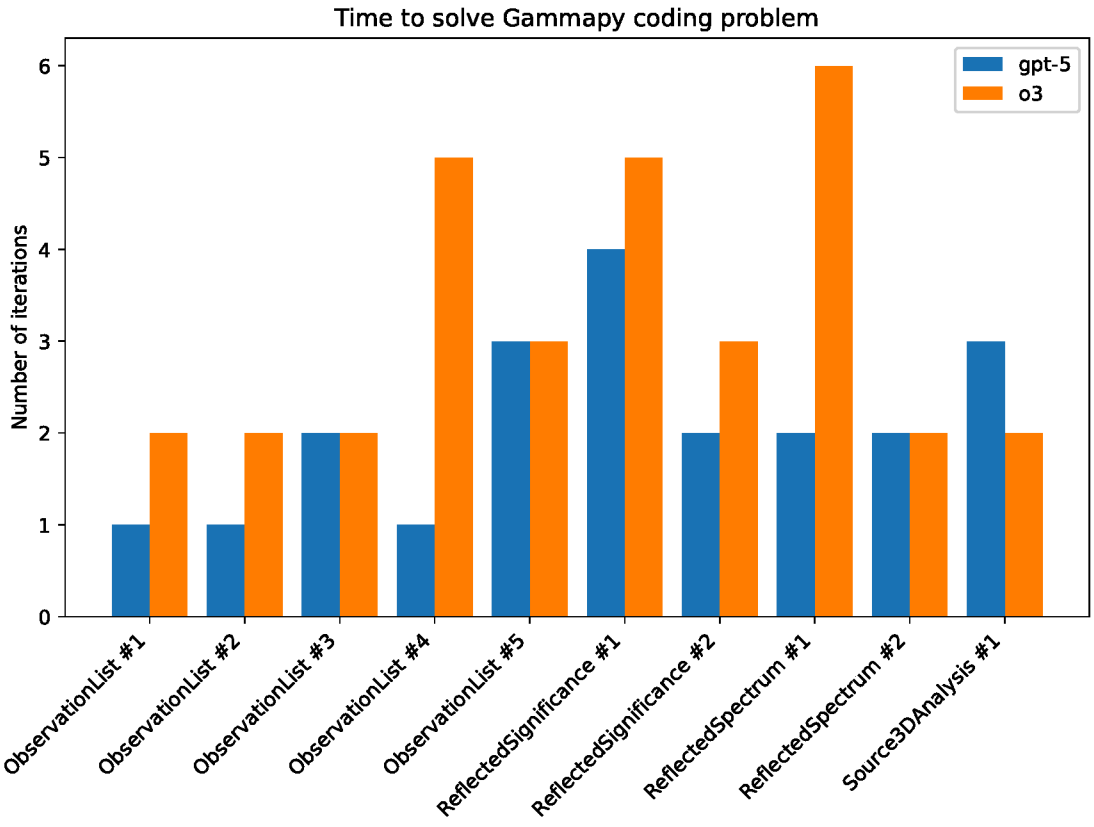

Figure 2: Coding benchmark results (attempts to pass and pass rates per task/model).

Results. On the smaller *per-source* tasks both models with high reasoning effort reached 100% pass rate in our runs, the most recent model shows slightly faster performance. The histogram in Figure 2 summarizes attempts-to-pass across tasks. As an illustration of trace logging, a typical successful run recorded 7.3k output tokens of which ~6.5k were reasoning tokens.

4. Conclusion

Domain-aware code agents are already practical for gamma-ray astronomy. By aligning a state-of-the-art reasoning model with up-to-date `Gammapy` workflows and wrapping it in a strict validation loop, our agent delivers reliable analysis scripts and reduces boilerplate for routine tasks.

In parallel, we are extending the backend layer to support *open-weight* models for on-premise and privacy-preserving deployments. Concretely, we are operating and evaluating Qwen [8] and OpenAI's GPT-OSS [9] series on the Helmholtz *Blablador* platform[5][6], which exposes an OpenAI-compatible API and allows researchers to run models locally under institutional control. This makes the agent agnostic to the hosting model—cloud or on-prem—and enables systematic comparisons between proprietary and open-weight backends in our benchmark harness.

Future work includes expanding the RAG corpus (e.g. CTAO simulations) for end-to-end sensitivity studies, exploring multi-agent collaboration (planner, critic, optimiser) to improve robustness and convergence, and hardening open-weight deployments (quantization, schedulers, and multi-GPU inference) on Blablador. We plan to continue releasing the package and benchmarks to support transparent, reproducible research across both closed and open ecosystems.

Acknowledgements

This work makes use of data from the H.E.S.S. DL3 public test data release 1, analyzed with the Gammapy framework. We thank Alexandre Strube for his assistance with the Blablador platform. The proceedings text is an experimental attempt at building an agent for automatic prose generation based on research results.

References

[1] W.X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou et al., *A survey of large language models*, 2303.18223.

[2] J. Luo, W. Zhang, Y. Yuan, Y. Zhao, J. Yang, Y. Gu et al., *Large Language Model Agent: A Survey on Methodology, Applications and Challenges*, *arXiv e-prints* (2025) arXiv:2503.21460 [2503.21460].

[3] J. Wei, Y. Yang, X. Zhang, Y. Chen, X. Zhuang, Z. Gao et al., *From AI for Science to Agentic Science: A Survey on Autonomous Scientific Discovery*, *arXiv e-prints* (2025) arXiv:2508.14111 [2508.14111].

[5] https://strube1.pages.jsc.fz-juelich.de/2024-02-talk-lips-blablador/
[6] https://sdlaml.pages.jsc.fz-juelich.de/ai/guides/blablador_api_access/

[4] D. Kostunin, V. Sotnikov, S. Golovachev and A. Strube, *AI Agents for Ground-Based Gamma Astronomy*, *arXiv e-prints* (2025) arXiv:2503.00821 [2503.00821].

[5] A. Donath, R. Terrier, Q. Remy, A. Sinha, C. Nigro, F. Pintore et al., *Gammapy: A Python package for gamma-ray astronomy*, *A&A* **678** (2023) A157.

[6] H.E.S.S. collaboration, *H.E.S.S. first public test data release*, 1810.04516.

[7] OpenAI, *GPT-5 System Card*, Tech. Rep. https://cdn.openai.com/gpt-5-system-card.pdf.

[8] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng et al., *Qwen3 Technical Report*, *arXiv e-prints* (2025) arXiv:2505.09388 [2505.09388].

[9] OpenAI, :, S. Agarwal, L. Ahmad, J. Ai, S. Altman et al., *gpt-oss-120b & gpt-oss-20b Model Card*, *arXiv e-prints* (2025) arXiv:2508.10925 [2508.10925].