Automating High Energy Physics Data Analysis with LLM-Powered Agents

Haichen Wang
UC Berkeley
haichenwang@berkeley.edu

Joshua Ho
UC Berkeley
ho22joshua@berkeley.edu

Luc Tomas Le Pottier
UC Berkeley
luclepot@berkeley.edu

Dongwon Kim
UC Berkeley
dwkim@berkeley.edu

Chengxi Yang
UC Berkeley
cxyang@berkeley.edu

Eli Gendreau-Distler
UC Berkeley
egendreaudistler@berkeley.edu

Abstract

We present a proof-of-principle study demonstrating the use of large language model (LLM) agents to automate a representative high energy physics (HEP) analysis. Using the Higgs boson diphoton cross-section measurement as a case study with ATLAS Open Data, we design a hybrid system that combines an LLM-based supervisor–coder agent with the `Snakemake` workflow manager. In this architecture, the workflow manager enforces reproducibility and determinism, while the agent autonomously generates, executes, and iteratively corrects analysis code in response to user instructions. We define quantitative evaluation metrics – success rate and error distribution – to assess agent performance across multi-stage workflows. To characterize variability across architectures, we benchmark a representative selection of state-of-the-art LLMs, spanning the *Gemini* and *GPT-5* series, the *Claude* family, and leading open-weight models. While the workflow manager ensures deterministic execution of all analysis steps, the final outputs still show stochastic variation. Although we set the temperature to zero, other sampling parameters (e.g., top-p, top-k) remained at their defaults, and some reasoning-oriented models internally adjust these settings. Consequently, the models do not produce fully deterministic results. This study establishes the first LLM-agent-driven automated data-analysis framework in HEP, enabling systematic benchmarking of model capabilities, stability, and limitations in real-world scientific computing environments. The baseline code used in this work is available at https://huggingface.co/HWresearch/LLM4HEP/tree/main.

1 Introduction

While large language models (LLMs) and agentic systems have been explored for automating components of scientific discovery in fields such as biology (e.g., CellAgent for single-cell RNA-seq analysis [Xiao et al., 2024]) and software engineering (e.g., LangGraph-based bug fixing agents [Wang and Duan, 2025]), their application to high energy physics (HEP) remains largely unexplored. In HEP, analysis pipelines are highly structured, resource-intensive, and with strict requirements of reproducibility and statistical rigor. Existing work in HEP has primarily focused on machine learning models for event classification or simulation acceleration, and more recently on domain-adapted LLMs (e.g., FeynTune [Richmond et al., 2025], Astro-HEP-BERT [Simons, 2024], BBT-Neutron [Wu et al., 2024]) and conceptual roadmaps for large physics models. However, to the best of our knowledge, no published study has demonstrated an agentic system that autonomously generates, executes, validates, and iterates on HEP data analysis workflows.

Machine Learning and the Physical Sciences Workshop, NeurIPS 2025.

In this work, we present the first attempt to operationalize LLM-based agents within a reproducible HEP analysis pipeline. Our approach integrates a task-focused agent with the `Snakemake` workflow manager [Mölder et al., 2021], leveraging `Snakemake`'s HPC-native execution and file-based provenance to enforce determinism, while delegating bounded subtasks (e.g., event selection, code generation, validation, and self-correction) to an LLM agent. This design departs from prior agent frameworks such as LangChain [Chase, 2022] or LangGraph [Team, 2025] that emphasize flexible multi-step reasoning by embedding the agent within a domain-constrained directed acyclic graph (DAG), ensuring both scientific reliability and AI relevance.

2 Description of a representative high energy physics analysis

In this paper, we use a cross-section measurement of the Higgs boson decaying to the diphoton channel at the Large Hadron Collider as an example. We employ collision data and simulation samples from the 2020 ATLAS Open Data release [ATLAS Collaboration, 2020]. The data sample corresponds to $10\,\mathrm{fb}^{-1}$ of proton-proton collisions collected in 2016 at $\sqrt{s} = 13$ TeV. Higgs boson production is simulated for gluon fusion, vector boson fusion, associated production with a vector boson, and associated production with a top-quark pair. Our example analysis uses control samples derived from collision data and Higgs production simulation samples to design a categorized analysis with a machine-learning-based event classifier, similar to the one used by the CMS collaboration in their Higgs observation paper [CMS Collaboration, 2012]. The objective of this analysis is to maximize the expected significance of the Higgs-to-diphoton signal. Since the purpose of this work is to demonstrate the LLM-based agentic implementation of a HEP analysis, we do not report the observed significance. The workflow of this analysis is highly representative in HEP.

The technical implementation of the analysis workflow is factorized into five sequential steps, executed through the `Snakemake` workflow management system. Each step is designed to test a distinct type of reasoning or code-generation capability, and the evaluation of each step is performed independently, i.e., the success of a given step does not depend on the completion or correctness of the previous one. This design allows for consistent benchmarking across heterogeneous tasks while maintaining deterministic workflow execution.

Step 1 (**ROOT file inspection**) generates summary text files describing the ROOT files and their internal structure, including trees and branches, to provide the agent with a human-readable overview of the available data.

Step 2 (**Ntuple conversion**) produces a Python script that reads all particle observables specified in the user prompt from the `TTree` objects in ROOT files [Antcheva et al., 2009] and converts them into `numpy` arrays for downstream analysis.

Step 3 (**Preprocessing**) normalizes the signal and background arrays and applies standard selection criteria to prepare the datasets for machine-learning–based classification.

Step 4 (**S-B separation**) applies TabPFN [Hollmann et al., 2025], a transformer-based foundation model for tabular data, to perform signal–background separation. The workflow requires a script that calls a provided function to train and evaluate the TabPFN model using the appropriate datasets and hyperparameters.

Step 5 (**Categorization**) performs statistical categorization of events by defining optimized boundaries on the TabPFN score to maximize the expected significance of the Higgs-to-diphoton signal. This step is implemented as an iterative function-calling procedure, where new category boundaries are added until the improvement in expected significance falls below 5%.

3 Architecture

We adopt a hybrid approach to automate the Higgs boson to diphoton data analysis. Given the relatively fixed workflow, we use `Snakemake` to orchestrate the sequential execution of analysis steps. A supervisor–coder agent is deployed to complete each step. This design balances the determinism of the analysis structure with the flexibility often required in HEP analyses.

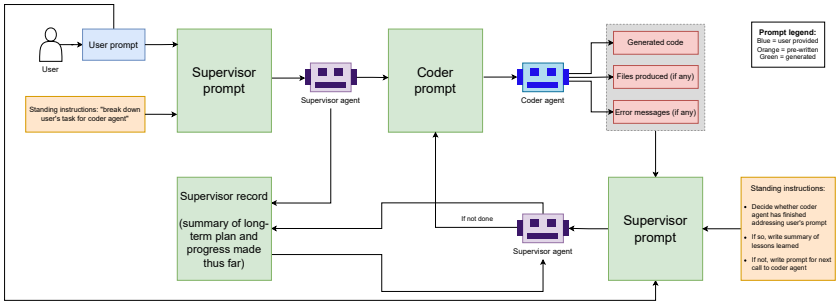


Figure 1: Illustration of internal workflow for the supervisor-coder agent.

3.1 Workflow management

`Snakemake` is a Python-based workflow management system that enables researchers to define computational workflows through rule-based specifications, where each rule describes how to generate specific output files from input files using defined scripts or commands. Snakemake serves as the workflow orchestration engine that manages the complex dependencies and execution order of the HEP analysis pipeline. In this package, `Snakemake` operates as the backbone that coordinates the five-stage analysis workflow for ATLAS diphoton data processing. This modular approach allows the complex physics analysis to be broken down into manageable, interdependent components that can be executed efficiently and reproducibly.

3.2 Supervisor-coder agent

We design a supervisor-coder agent to carry out each task, as illustrated in Fig. 1. The supervisor and coder are implemented as API calls to a LLM, with distinct system prompts tailored to their respective roles. The supervisor receives instructions from the human user, formulates corresponding directives for the coder, and reviews the coder's output. The coder, in turn, takes the supervisor's instructions and generates code to implement the required action. The generated code is executed through an execution engine, which records the execution trace. The supervisor and coder roles are defined by their differing access to state, memory, and system instructions. In the reference configuration, both roles are implemented using the `gemini-pro-2.5` Google [2025] model; however, the same architecture is applied to a range of contemporary LLMs to evaluate model-dependent performance and variability.

Each agent interaction is executed through API calls to the LLM. Although we set the temperature to 0, other sampling parameters (e.g., top-p, top-k) remained at their default values, and some thinking-oriented models internally raise the effective temperature. As a result, the outputs exhibit minor stochastic variation even under identical inputs. Each call includes a user instruction, a system prompt, and auxiliary metadata for tracking errors and execution records.

For the initial user interaction with the supervisor, the input prompt includes a natural-language description of the task objectives, suggested implementation strategies, and a system prompt that constrains the behavior of the model output. The result of this interaction is an instruction passed to the coder, which in turn generates a Python script to be executed. If execution produces an error, the error message, the original supervisor instruction, the generated code, and a debugging system prompt are passed back to the supervisor in a follow-up API call. The supervisor then issues revised instructions to the coder to address the problem. This self-correction loop is repeated up to three times before the trial is deemed unsuccessful.

4 Results

To establish a baseline, we conducted 219 experiments with the `gemini-pro-2.5` model, providing high statistical precision across all analysis stages. In this configuration, the workflow was organized into three composite steps: (1) **data preparation** including *ntuple creation*, and *preprocessing*; (2) **signal–background (S-B) separation**; and (3) **categorization** based on the expected Higgs-to-photon

significance. Up to five self-correction iterations were applied. The corresponding success rates for these three steps were $58 \pm 3\%$, $88 \pm 2\%$, and $74 \pm 3\%$, respectively.

As summarized in Table 1, the **data-preparation** stage (combining ntuple production and preprocessing) was the most error-prone, with 93 failures out of 219 trials. Most issues stemmed from insufficient context for identifying objects within ROOT files. The dominant failure modes were Type 1 (zero event weights) and Type 5 (missing intermediate files), indicating persistent challenges in reasoning about file structures, dependencies, and runtime organization. Providing richer execution context such as package lists, file hierarchies, and metadata could help mitigate these problems. Despite these limitations, the overall completion rate ($\sim 57\%$) demonstrates that LLMs can autonomously execute complex domain-specific programming tasks a non-trivial fraction of the time.

The subsequent **S-B separation** using `TabPFN` and the final **categorization** for the Higgs-to-diphoton measurement exhibited substantially fewer failures (25 and 57, respectively). These errors were primarily Type 3 (function-calling) and Type 6 (semantic) issues, reflecting improved stability once the workflow reaches the model-training and evaluation stages.

Step	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6	Type 7
Data-preparation (93 failures)	41	15	1	9	17	6	4
S-B separation (25 failures)	0	4	3	0	13	5	0
Categorization (57 failures)	0	4	26	0	6	17	4

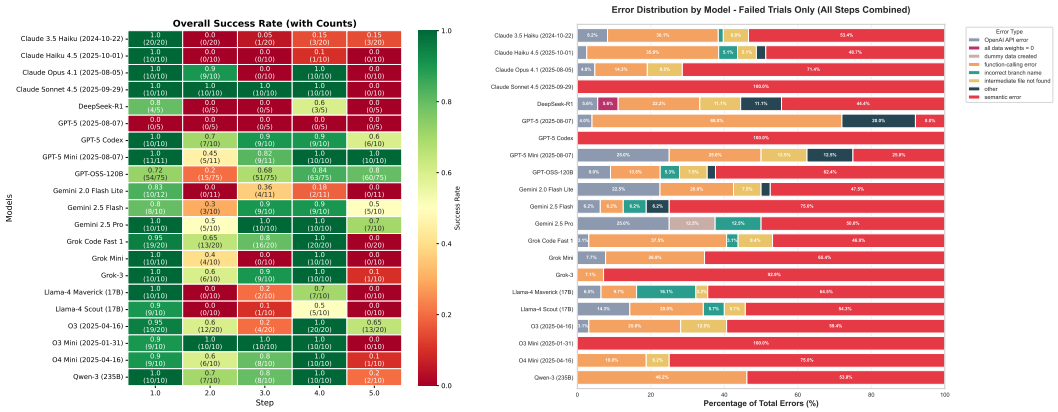
Table 1: Distribution of failure counts by error category for each workflow step, based on 219 trials per step. *Error type definitions:* Type 1 — all data weights = 0; Type 2 — dummy data created; Type 3 — function-calling error; Type 4 — incorrect branch name; Type 5 — intermediate file not found; Type 6 — semantic error; Type 7 — other.

To assess agent efficiency, we measured the ratio of user-input tokens to the total tokens exchanged with the API. For the `gemini-pro-2.5` baseline, this ratio was $(1.65 \pm 0.15) \times 10^{-3}$ for **data preparation**, $(1.43 \pm 0.10) \times 10^{-3}$ for **S-B separation**, and $(0.93 \pm 0.07) \times 10^{-3}$ for **categorization**. A higher ratio indicates more efficient task execution, as fewer internally generated tokens are required to complete the same instruction set. Over 98% of tokens originated from the model's autonomous reasoning and self-correction rather than direct user input, suggesting that additional task detail would minimally affect overall token cost. This ratio thus provides a simple diagnostic of communication efficiency and reasoning compactness.

Following the initial benchmark with `gemini-pro-2.5`, we expanded the study to include additional models, such as `openai-gpt-5` [OpenAI, 2025], `claude-3.5` [Anthropic, 2024], `qwen-3` [Alibaba, 2025], and the open-weight `gpt-oss-120b` [OpenAI et al., 2025], evaluated under the same agentic workflow. Based on early observations, the prompts for the **data preparation** stage were refined and divided into three subtasks: **ROOT file inspection**, **ntuple conversion**, and **preprocessing** (signal and background region selection). Input file locations were also made explicit for an agent to ensure deterministic resolution of data paths and reduce reliance on implicit context.

The results across models, summarized in Figs. 2a–2b, show consistent qualitative behavior with the baseline while highlighting quantitative differences in reliability, efficiency, and error patterns. For the `gemini-pro-2.5` model, the large number of repeated trials (219 total) provides a statistically robust characterization of performance across steps. For the other models - each tested with approximately ten trials per step - the smaller sample size limits statistical interpretation, and the results should therefore be regarded as qualitative indicators of behavior rather than precise performance estimates. Nonetheless, the observed cross-model consistency and similar failure patterns suggest that the workflow and evaluation metrics are sufficiently general to support larger-scale future benchmarks. This pilot-level comparison thus establishes both the feasibility and reproducibility of the agentic workflow across distinct LLM architectures.

Figure 2a summarizes the cross-model performance of the agentic workflow. The heatmap shows the success rate for each model–step pair, highlighting substantial variation in reliability across architectures. The dates appended to each model name denote the model release or update version used for testing, while the parameter in parentheses (e.g., "17B") indicates the model's approximate parameter count in billions. Models in the `GPT-5` and `Gemini` families achieve the highest completion fractions across most steps, whereas smaller or open-weight models such as `gpt-oss-120b` exhibit lower but still non-negligible success on certain steps. This demonstrates that the workflow can be



(a) Success rate

(b) Error distribution

Figure 2: Cross-model comparison of LLM-agent performance. (a) Success fraction for each model–step pair. (b) Distribution of error magnitudes by model, summarizing the variability and characteristic failure patterns across models. Bars without numerical labels correspond to fractions below 3%, omitted for clarity.

executed end-to-end by multiple LLMs, though with markedly different robustness and learning stability.

Figure 2b shows the distribution of failure modes across all analysis stages, considering only trials that did not reach a successful completion. The categorization is based on the LLM-generated outputs themselves, capturing how each model typically fails. Error types include function-calling issues, missing or placeholder data, infrastructure problems (e.g., API or file-handling failures), and semantic errors - cases where the model misinterprets the prompt and produces runnable but incorrect code, as well as syntax mistakes.

Clear model-dependent patterns emerge. Models with higher success rates, such as the `GPT-5` and `Gemini 2.5` families, show fewer logic and syntax issues, while smaller or open-weight models exhibit more semantic and data-handling failures. These differences reveal characteristic failure signatures that complement overall completion rates.

Taken together, Figure 2 highlights two main aspects of LLM-agent behavior: overall task reliability and the characteristic error modes behind failed executions. These results show that models differ not only in success rate but also in the nature of their failures, providing insight into their robustness and limitations in realistic HEP workflows.

5 Limitations

This study shows that LLMs can support HEP data analysis workflows by interpreting natural language, generating executable code, and applying basic self-correction. While multi-step task planning is beyond the current scope, the `Snakemake` integration provides a natural path toward rule-based agent planning. Future work will pursue this direction and further strengthen the framework through improvements in prompting, agent design, domain adaptation, and retrieval-augmented generation.

6 Conclusion

We demonstrated the feasibility of employing LLM agents to automate components of high-energy physics (HEP) data analysis within a reproducible workflow. The proposed hybrid framework combines LLM-based task execution with deterministic workflow management, enabling near end-to-end analyses with limited human oversight. While the `gemini-pro-2.5` model served as a statistically stable reference, the broader cross-model evaluation shows that several contemporary LLMs can perform complex HEP tasks with distinct levels of reliability and robustness. Beyond this proof of concept, the framework provides a foundation for systematically assessing and improving LLM-agent performance in domain-specific scientific workflows.

References

Yihang Xiao, Jinyi Liu, Yan Zheng, Xiaohan Xie, Jianye Hao, Mingzhi Li, Ruitao Wang, Fei Ni, Yuxiao Li, Jintian Luo, Shaoqing Jiao, and Jiajie Peng. CellAgent: An LLM-driven Multi-Agent Framework for Automated Single-cell Data Analysis, 2024. URL https://arxiv.org/abs/2407.09811.

Jialin Wang and Zhihua Duan. Empirical Research on Utilizing LLM-based Agents for Automated Bug Fixing via LangGraph, 2025. URL https://arxiv.org/abs/2502.18465.

Paul Richmond, Prarit Agarwal, Borun Chowdhury, Vasilis Niarchos, and Constantinos Papageorgakis. FeynTune: Large Language Models for High-Energy Theory, 2025. URL https://arxiv.org/abs/2508.03716.

Arno Simons. Astro-HEP-BERT: A bidirectional language model for studying the meanings of concepts in astrophysics and high energy physics, 2024. URL https://arxiv.org/abs/2411.14877.

Hengkui Wu, Panpan Chi, Yongfeng Zhu, Liujiang Liu, Shuyang Hu, Yuexin Wang, Chen Zhou, Qihao Wang, Yingsi Xin, Bruce Liu, Dahao Liang, Xinglong Jia, and Manqi Ruan. Scaling Particle Collision Data Analysis, 2024. URL https://arxiv.org/abs/2412.00129.

Felix Mölder, Kim Philipp Jablonski, Benjamin Letcher, et al. Sustainable data analysis with snakemake. *F1000Research*, 10:33, 2021. doi: 10.12688/f1000research.29032.1. URL https://doi.org/10.12688/f1000research.29032.1. [version 1; peer review: 1 approved, 1 approved with reservations].

Harrison Chase. LangChain: A Framework for Large Language Model Applications, 2022. URL https://langchain.com/. Open-source software framework for integrating LLMs; Python & JavaScript.

LangChain Team. LangGraph: A Graph-Based Agent Workflow Framework, 2025. URL https://langchain-ai.github.io/langgraph/. Open-source orchestration framework built on LangChain.

ATLAS Collaboration. ATLAS simulated samples collection for jet reconstruction training, as part of the 2020 Open Data release. CERN Open Data Portal, 2020. URL http://doi.org/10.7483/OPENDATA.ATLAS.L806.5CKU.

CMS Collaboration. Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC. *Phys. Lett. B*, 716:30–61, 2012. doi: 10.1016/j.physletb.2012.08.021.

I. Antcheva et al. ROOT: A C++ framework for petabyte data storage, statistical analysis and visualization. *Comput. Phys. Commun.*, 180:2499–2512, 2009. doi: 10.1016/j.cpc.2009.08.005.

N. Hollmann, S. Müller, L. Purucker, et al. Accurate predictions on small data with a tabular foundation model. *Nature*, 637:319–326, 2025. doi: 10.1038/s41586-024-08328-6. URL https://doi.org/10.1038/s41586-024-08328-6.

Google. Gemini 2.5 Pro, August 2025. URL https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-pro. Accessed 2025-08-29.

OpenAI. GPT-5, 2025. URL https://platform.openai.com/docs/models/gpt-5. Accessed 2025-08-29.

Anthropic. Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet, 2024. Accessed: 2025-11-11.

Alibaba. Qwen3, April 2025. URL https://qwenlm.github.io/blog/qwen3/. Accessed 2025-08-29.

OpenAI et al. gpt-oss-120b gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

ATLAS Collaboration. Observation of Higgs boson production in association with a top quark pair at the LHC with the ATLAS detector. *Phys. Lett. B*, 784:173–191, 2018. doi: 10.1016/j.physletb.2018.07.035.

A Technical Appendices and Supplementary Material

A.1 Features used for TabPFN classifier

The language model is asked to save an extensive list of particle observables that would be useful for a more sophisticated physics analysis: the transverse momentum (p_T), pseudorapidity (η), and azimuthal angle (ϕ) of the two leading photons, of the two leading leptons, and of the six leading jets; the transverse momentum and azimuthal angle of the missing transverse energy (MET); the Monte Carlo weight; the tight photon ID flags; the cross section; and the sum of Monte Carlo weights. However, in the interest of efficiency we use only the photon observables for the TabPFN classifier. The photon p_T's are normalized by the invariant mass to prevent the classifier from learning to separate the signal and background events by the invariant mass. In addition, because only differences in azimuthal angle are physically meaningful, we replace the two diphoton ϕ coordinates with their difference $\phi_1 - \phi_2$.

A.2 Event selection, training sample definition, and background estimation

Preselection To roughly follow the expected topology of a Higgs boson decaying into two photons, photon candidates are taken from well-calibrated regions of the detector ($|\eta| < 2.37$), while the transition zone between the barrel and endcap ($1.37 < |\eta| < 1.52$), where measurements are less reliable, is set aside. Each photon is required to have transverse energy above 25 GeV, ensuring that selected photons are energetic enough to be measured with good resolution and to suppress backgrounds from softer processes. Events are kept if they contain two photons that meet identification and isolation criteria, which help distinguish genuine photons from misidentified particles or nearby activity. To reflect the expected kinematics of Higgs decays, the leading photon must contribute more than 35% of the diphoton system's momentum ($p_T/m_{\gamma\gamma} > 0.35$), and the sub-leading photon more than 25%. Finally, events are restricted to a diphoton invariant mass window of $105 < m_{\gamma\gamma} < 160$ GeV, a range chosen to capture the Higgs boson signal region while suppressing background contributions at much lower or higher masses.

Signal and background classifier training sample Following the strategy of Ref. ATLAS Collaboration [2018], events in which at least one photon does not meet the tight identification requirement are used to model the background in the signal region. These so-called NTI events are taken from the diphoton invariant mass sidebands, 105–120 GeV and 130-160 GeV, ranges chosen to minimize potential Higgs boson contributions. Because NTI events exhibit kinematic properties similar to those of tightly identified and isolated photons, they serve as a reliable proxy for background processes. For the signal sample, we use simulated Higgs boson events in which both photons satisfy the tight identification and isolation requirements. To ensure that the training is not biased toward either sample, signal and background events are rescaled to have equal weights.

Background estimate for sensitivity evaluation To estimate the number of background events in the signal region (where both photons satisfy the tight isolation requirement), we rely on NTI events as a control sample. The signal is expected to appear as a narrow peak in the diphoton invariant mass distribution, so the significance is evaluated within a small mass window around the Higgs boson, 125 ± 2 GeV. The expected background in this window is inferred from the NTI sidebands using

$$\text{Expected background} = SF1 \times SF2 \times NTI_{\text{yield}}.$$

In this expression, NTI_{yield} denotes the number of NTI data events observed in the sideband regions. The scale factor $SF1$ accounts for the difference in yields between tight-isolated (TI) and NTI photons in the sidebands, while $SF2$ adjusts for the relative abundance of NTI events in the signal window compared with the sidebands. In this way, the measured background in the sidebands is extrapolated into the Higgs signal region.

A.3 Procedure of event categorization

The goal of the event categorization is to select boundaries among 1000 evenly spaced values of the signal-background separation score in a way that maximizes the statistical significance of the

Higgs-to-diphoton signal process. We adopt a number counting significance formula given by

$$Z_i = \sqrt{2\left[(s_i + b_i)\ln\left(1 + \frac{s_i}{b_i}\right) - s_i\right]}, \quad Z_{\text{tot}} = \sqrt{\sum_i Z_i^2}$$

where summation runs over all categories and s_i/b_i denote numbers of signal/background events in i^{th} category. To maximize this significance, the first step is to scan over all possible locations for the boundary, compute the significance with each candidate boundary, and ultimately keep the boundary that yields the highest statistical significance. This procedure is then repeated until the most recently added boundary improves the overall significance by less than 5%.

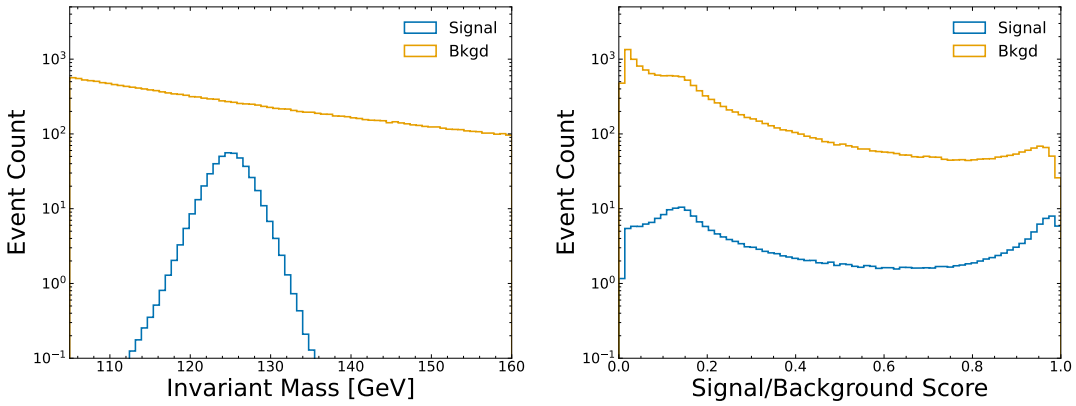


Figure 3: Left: Invariant mass distribution of signal and background processes after standard analysis selections are applied. Right: Signal-background separation scores generated by TabPFN. Note that the normalization is different in the left and right plots because we apply cuts on the invariant mass before computing the signal-background separation scores.