VESTA: Visual Exploration with Statistical Tool Agents

William Rudman[1,*], **Abhishek Divekar**[1,2*], **Kanishk Jain**[1*], **Sebastian Joseph**[1], **Stella S. R. Offner**[1], **Matthew Lease**[1], **Kyle Mahowald**[1], **Greg Durrett**[3†], **Junyi Jessy Li**[1†]

[1]The University of Texas at Austin [2]Amazon [3]New York University

Correspondence: william.rudman@utexas.edu | https://github.com/wrudman/VESTA

Abstract

Fitting quantitative models to data is a central step in scientific workflows, yet it remains one of the least automated. Recent agent-based systems leverage language and vision-language models (VLMs) to propose statistical models, but these systems struggle on challenging modeling tasks. To address these limitations, we introduce VESTA (**V**isual **E**xploration with **S**tatistical **T**ool **A**gents), a framework that equips VLMs with a dynamically growing toolkit to guide model refinement through data transformations, hypothesis-driven visualizations, and statistical tests. Unlike prior systems that rely on iterative critique alone, VESTA explores data before and during refinement by selecting or creating reusable diagnostic tools. We evaluate VESTA using three toolkit configurations: no tools, static expert-written tools, and dynamic model-written tools. To support this evaluation, we introduce DAWN (**D**ataset for **A**utomated **W**orkflows and **N**umerical Modeling), a benchmark targeting distribution fitting and time series modeling with varying difficulty tiers that culminates in real-world astronomy tasks including modeling initial mass functions and gravitational-wave chirp signals. VESTA's dynamic tool creation outperforms prior agentic pipelines for statistical analysis and visual tool creation, with the largest gains on complex and domain-specific tasks. Against strong agentic coding harnesses (Claude Code, Codex, and OpenCode), VESTA is better on all time series tasks and on complex distribution fitting tasks, though the harnesses achieve better fits on simpler distributions. Further, dynamically generated tools are substantially more sophisticated than those produced by existing visual tool-creation systems, covering more diagnostic categories per function and strongly preferring visual outputs that the VESTA VLM critic can reason over directly.

1 Introduction

Large Language Model (LLM) and Vision-Language Model (VLM) agents have demonstrated remarkable promise in scientific discovery, with systems capable of forming hypotheses, designing experiments, and synthesizing results. However, failures at different steps impede these models' abilities to truly accelerate science. This work focuses on the task of *fitting quantitative models to data*. This step is complex, requiring priors about the domain in question and careful statistical analysis, while being less studied than other quantitative analysis tasks like ML engineering and data science [10, 13]. Addressing this gap is particularly important in scientific disciplines like astronomy where models struggle to effectively use domain conventions [23].

We introduce VESTA[3] (**V**isual **E**xploration with **S**tatistical **T**ool **A**gents), a framework that proposes, refines, and evaluates statistical models by analyzing *visual representations* of data. VESTA operates

*Equal contribution.

†Equal advising.

[3]VESTA is the second most massive body in the main asteroid belt [32].

Preprint.

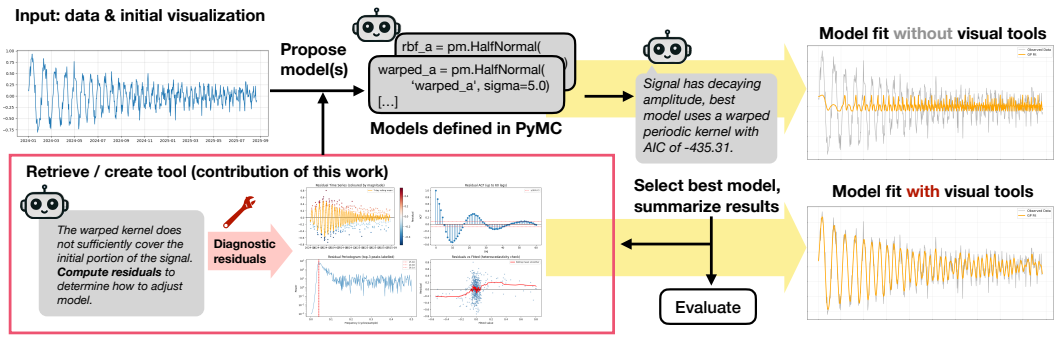


Figure 1: Overview of VESTA. By effectively using and creating tools, VESTA produces a probabilistic PyMC program that models the input data.

over models defined in PyMC [1], a Bayesian probabilistic programming framework that enables model definition and fitting to data using an efficient Markov Chain Monte Carlo (MCMC) method. Model-building with PyMC code is compositional, allowing for complex distributions to be constructed by combining simpler components. VESTA instantiates a loop of proposing models, selecting the best one, then iterating, broadly similar to existing autoresearch workflows [8, 40]. However, unlike prior work, VESTA utilizes multimodal feedback by augmenting this basic loop with a newer variety of **visual tools**. Visual tools allow VESTA to inspect model fits using a VLM, giving it leverage in a modality distinct from past work on tool use.

Figure 1 shows how a model can use a tool to compute residuals, produce a visualization, and then inspect that visualization to identify areas where the predicted model does not fit the data well and propose a new statistical model that better fits the data. For the diversity of statistical model fitting tasks we explore here, it is infeasible to produce all necessary tools beforehand; *dynamic* tool instantiation is required to cover the full breadth of cases.

To support advancement on challenging model fitting problems where current approaches fail, we introduce a new benchmark we call DAWN (**D**ataset for **A**utomated **W**orkflows and **N**umerical Modeling).[4] DAWN problems cover two domains central to both general data science and scientific research: fitting probability distributions to discrete data and constructing models for time-series analysis. In contrast to existing benchmarks which often contain a small number of distributions with a limited number of points [15, 29], DAWN consists of synthetic problems spanning two levels of difficulty (**Easy** and **Hard**), as well as two fundamental problems from astronomy (**Astro**): fitting stellar initial mass functions (IMFs), and detecting and characterizing chirp signals embedded in gravitational-wave data.

Using DAWN, we benchmark VESTA against prior frameworks specialized for Bayesian modeling and dynamic VLM-based tool-generation systems. While existing approaches and coding harnesses perform competitively on our **Easy** splits, performance degrades substantially as data complexity increases and tasks shift toward domain-specific scientific phenomena. This degradation reveals a systematic gap in current agents' capacity for iterative, evidence-driven model refinement. VESTA recovers expert-created tools at a higher rate than prior work, creating sophisticated tools that compose several expert tools, testing multiple hypotheses simultaneously. While dynamically generated visual tools do not outperform expert-written tools, likely due to current agents' limited ability to process complex diagrams. VESTA shows great promise at producing on-demand statistical tooling and we expect effectiveness will improve alongside rapidly evolving VLM capabilities.

2 Related Work

Automated Model Fitting Past work evaluated the capabilities of LLMs and VLMs to automate statistical analysis [42, 22], time-series modeling [52, 47, 16, 56, 34] and Bayesian model proposals [37, 28]. Ji et al. [22] and Sun et al. [42] survey how LLM-based agents can automate statistical workflows from data cleaning and model selection. However, Zhu et al. [59] find that even state-of-the-

[4] DAWN was NASA's first truly interplanetary spaceship that explored VESTA [33].

art LLMs achieve only modest accuracy on benchmarks for selecting appropriate statistical methods, revealing significant limitations in LLMs' ability to reason over complex statistical tasks. While much of the focus in the literature has used LLMs / VLMs to predict a continuation of time series or detect anomalies, our work seeks to represent the *underlying model* that generates the observed data using a probabilistic Bayesian model (PyMC). Previous work in generating PyMC code to model data has been successful in scientific settings, but this work often relies on human-written kernels for Gaussian processes, limiting the use cases of such methods [5, 6, 12]. Recent work addresses these key limitation by using LLMs to automatically perform "Box's Loop" of iteratively proposing new models and critiquing existing ones written in PyMC code until an adequate fit is achieved [27]. Despite some promising results using LLMs to propose probabilistic models, Gandhi et al. [15] find LLMs struggle on more challenging datasets sampled from real-world domains and demonstrate that the iterative refinement of PyMC models does not consistently provide better predictions over one-shot LLMs. Further, while existing benchmarks only cover a narrow range of distributions, DAWN combines a variety of complex mixture distributions inspired by real world tasks. Our VESTA automates Box's propose-critique-refine loop using VLMs, enabling tool use and creation for data analysis, while handling significantly more complex distributions.

Multimodal Tool Use & Creation Several works have demonstrated that VLMs are proficient tool-users and that the successful utilization of tools can improve fine-grained reasoning [20, 57, 48, 51]. Incorporating VLM tool-use in agentic systems has produced even larger performance gains [53, 18, 50, 19]. In particular, Zhang et al. [54] propose a multi-agent VLM framework in which an orchestrator agent coordinates specialized sub-agents and vision expert tools to improve fine-grained visual perception. In PyVision [55], VLMs are used in an agentic flow where they dynamically create and execute tools until an answer is reached. Previous work has limited VLM tool use (i.e., executing or generating code over visual inputs) to image enhancements, annotations, cropping, and the calculation of basic statistics [17, 44, 58]. While some work has applied multimodal tool creation to model fitting [24], such tools are limited to simple statistics and basic plotting, lack a dedicated tool-creation step, and are discarded after each use rather than persisted for reuse across iterations. In contrast, VESTA uses an LLM tool agent to dynamically create visualizations that test statistical hypotheses, which are then passed to a VLM critic as evidence for refining them.

Autonomous Tool Expansion While both LLMs and VLMs have proven to use tools effectively, finding the optimal set of tools for a problem often requires manually curating a toolkit. Initial work demonstrates that LLMs are capable of coding effective and reliable tools [36, 7], although these tools are typically created separately from model inference [2, 49]. Systems such as Voyager [46] extend this static paradigm by allowing for tool libraries to adapt to task needs and evolve across examples. Many of the systems for autonomous tool expansion tend to focus on software engineering tasks [11, 43, 26] or narrowly scoped API invocation benchmarks [21, 41], with less focus on problems that require judgment-driven statistical analysis. While EvoSkill [2] and TTE [30] create some tools related to calculating statistics, they tend to contain more simplistic functions such as calculating averages, areas or retrieving information from tables.

3 VESTA: Visual Exploration with Statistical Tools

We assume as input a dataset D generated by a model $M(\theta)$. M is a data generating process (e.g., a directed graphical model) with parameters θ. In this work, we consider models that can be specified in PyMC code. Given a proposed model $\tilde{M}(\tilde{\theta})$, we can evaluate its fit to D with metrics denoted as $R(D, \tilde{M}(\tilde{\theta}))$. Central to our approach are *visual tools*, $\varepsilon(D, M, \theta)$, implemented as executable Python functions that accept the data D and a current model (M, θ) and return visual or statistical diagnostics used to inform subsequent modeling decisions.

3.1 VESTA

Algorithm 1 illustrates the workflow for VESTA. We start with an initial prediction by estimating a probabilistic model from an initial plot of the data, and then refine this prediction over N steps. At each step, we propose p alternative candidate models, then select the model with the best metric, R. Our approach proceeds in four steps per iteration: propose, generate tools, critique model fit, and summarize steps of the run.

Propose. On the first iteration, VESTA takes an initial visualization of the data, produces a brief visual description and is instructed to create p diverse model proposals. To ensure faithfulness to the proposed model parameters, PyMC code is generated in a dedicated step that translates parameter specifications into executable PyMC programs. Without this step, we find models frequently deviate from their specified parameterizations.

Generate Tools. The central contribution of VESTA is its ability to explore data through the composition and creation of tools. At each iteration, VESTA receives a summary of the previous runs then decides to either select an existing tool from a growing registry $\mathcal{E}$, or dynamically codes a new tool in Python when no existing tool is adequate for the desired analysis. Newly created tools are saved to the registry, making them reusable for the current run. At each run, the toolkit is newly initialized. Section 6.2 presents a case study of generated tools (Figure 5).

Algorithm 1 VESTA

Require: Data D, iterations N, proposals p, metric R, registry $\mathcal{E}$ (initial state: `generate_new_tool` only)
Ensure: $M_{\text{best}}, \theta_{\text{best}}$

$M_{\text{best}}, \theta_{\text{best}} \leftarrow \textsc{Propose}(\text{Plot}(D))$ ▷ Plot data; fit j models and select best model under metric, R
$s_0 \leftarrow \textsc{Summarize}(M_{\text{best}}, \theta_{\text{best}})$
for $i = 1, \ldots, N$ **do**
  $\varepsilon^* \leftarrow \textsc{ToolManager}(s_{i-1}, D, \mathcal{E})$ ▷ Select/create tool; update $\mathcal{E}$
  $T_i \leftarrow \varepsilon^*(D, M_{i-1}, \theta_{i-1})$ ▷ Execute tool
  $M_i, \theta_i \leftarrow \textsc{Critique}(T_i, s_{0:i-1}, D)$ ▷ Propose p models; return best ranked by R
  $M_{\text{best}}, \theta_{\text{best}} \leftarrow \arg\min_{\theta \in \{\theta_i, \theta_{\text{best}}\}} R(\mathcal{M}(\theta))$
  $s_i \leftarrow \textsc{Summarize}(M_i, \theta_i)$
end for
return $M_{\text{best}}, \theta_{\text{best}}$

Critique. The output of the selected or newly created tool, along with the history of the best models and descriptions from previous runs is passed to a VLM critic that refines the current PyMC model by proposing p revised models. The revised model that optimizes R is passed to the next iteration. In our prompt, we instruct the VLM critic to terminate the iterative refinement loop early if it judges the current model fit to be satisfactory. The critic signals this with a `COMPLETE` token.

Summarize. At the end of each iteration, the visual description, chosen model and parameters, the metric R, and tool output are compressed into a structured summary. This summary is injected into the next iteration in place of raw intermediate outputs, enabling the system to reason over the full refinement trajectory without incurring prohibitive context growth.

3.2 Visual Tools in VESTA

VESTA can work out-of-the-box without tools. However, we found tools to be critical in solving challenging model fitting tasks. In particular, we allow VESTA to propose new tools tailored to its model predictions. At each tool generation step, VESTA receives a summary of the previous iteration, including the VLM's critique of the model fit, the best PyMC model and its score $R(D, M_{\text{best}}(\theta))$, and a history of previously executed tools. The agent also has access to a registry of available tools. We initialize the toolkit with a single function that gives the option for creating a new tool. New tools emerge through the process of examining data, and allow models to verify and update an existing hypothesis (e.g., a proposed statistical distribution). We save successfully created tools. If a generated tool fails, we allow up to two retries, passing the failure trace into the retry prompt. After two retries if a tool fails to execute due to syntax or import errors, the system falls back to the "no toolkit" mode.

In practice, VESTA generates complex tools that test distinct hypotheses through visualization, and the Critique stage uses their outputs (case study in Section 6.2, Figure 5). VESTA can also use externally provided tools. Section 5 describes our **expert-written** toolkit, which we evaluate in Section 6.

4 The DAWN Benchmark

Distribution fitting and time series modeling are two key data science modeling tasks that appear consistently across scientific disciplines. We select these domains because they allow us to benchmark AI agents across easily recognizable models, complex mixtures of common forms, and challenging

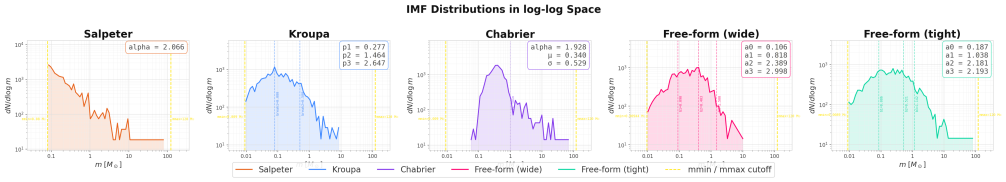


Figure 3: DAWN's **Astro** distribution fitting tasks. Example of Initial Mass Functions projected into log-log space. Distributions become visually distinct only when projected into log-log space.

real-world tasks where domain-specific knowledge, such as established conventions and governing equations, can reduce the complexity of modeling real-world phenomena. DAWN problems are synthetically generated, allowing us to robustly measure model fit.

4.1 Distribution Fitting Tasks

The **Easy** tasks in distribution fitting consist of identifying the family and associated parameters for a unimodal distribution. The distribution families we consider in this paper are: `gaussian`, `lognormal`, `student-t`, `exponential`, `uniform`, `weibull`, `laplace`, `cauchy` and `pareto`. Both the priors and exact number of points for each distribution are randomly sampled. The **Hard** task composes exactly two of these distributions. Mixtures present in the **Hard** split model phenomenon present in other real-world domains such as the gaussian-laplace mixtures common in speech signal processing [3] and lognormal-exponential mixtures prevalent in survival-analysis [31].

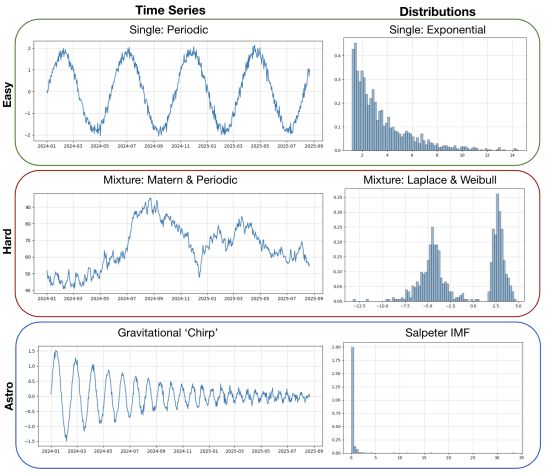


Figure 2: Sample inputs from both domains and all dataset splits in DAWN. **Easy** splits contain easily recognizable forms. **Hard** tasks contain mixtures of distinct forms, and **Astro** tasks reflect real-world astronomy challenges that require additional analysis beyond simple visualization to solve.

For our **Astro** split, we use initial mass functions (IMF) as the distribution family, which describe the distribution of stellar masses at birth and are often modeled with canonical forms [35]. Variations in its form offer a window into past star-forming environments. Because the IMF is derived empirically, accurate statistical modeling is critical to scientific inference. Accurate modeling of the IMF allows astronomers to infer the conditions of past star formation and predict its downstream effects on galactic evolution and heavy element distribution [4].

We use five functional forms (visualized in Figure 3): **(1)** The simplest form, the Salpeter IMF [39], is a single power law. **(2)** The Kroupa IMF [25] is a piecewise power law with sharp breaks at specific characteristic masses. **(3)** The Chabrier IMF [9] is distinguished by a smooth log-normal component at low masses transitioning to a power law at higher masses. We additionally include two freeform variants that do not follow canonical forms, making them more challenging to model: **(4)** one with breaks constrained near canonical masses (freeform tight) and **(5)** one with unconstrained breaks and slopes that can produce more exotic stellar IMFs (freeform wide).

Evaluation Metric. We use Jensen-Shannon Divergence to measure similarity between our proposed model's distribution and the reference model's distribution. This metric is symmetric and bounded in the interval $[0, 1]$, with a score of 0 indicating identical distributions. Additionally, in Appendix E.5 we analyze the impact of alternative evaluation metrics and use a "slope parameter recovery" metric specifically designed for IMF fitting.

4.2 Time Series Fitting Tasks

The **Easy** task in time series fitting consists of modeling sequences characterized by simple linear trends (increasing, decreasing, or flat) and standard seasonal components, such as basic periodic sine waves, coupled with minimal noise. The **Hard** task introduces more complex, non-linear dynamics and non-standard periodicities. This split includes time series exhibiting random walks (ARIMA process), and S-curve (sigmoidal) functions. Further, **Hard** contains time series from other domains, such as electrocardiogram (ECG) signals of heartbeat patterns and time series exhibiting seasonal periodicity with increasing averages common in sales forecasting [14]. Capturing the behavior of these series requires moving beyond standard linear or single-period functions to employ multi-periodic modeling and Gaussian process kernels. This split tests the model's ability to handle richer structures and moderate levels of noise and non-linearity.

For our **Astro** split, we model time series inspired by gravitational wave chirps. "Chirps" are an astronomical phenomenon produced by merging binary star systems that introduces severe non-stationarity into the signal. The standard chirp variant features a continuously changing frequency over time, where initial high-wavelength, low-frequency waves progressively compress into higher frequencies as the celestial bodies converge. We additionally include a highly complex variant where this dynamic, shifting frequency is coupled with an amplitude that decays over time.

Evaluation Metric. Similar to prior works [27], we use the Expected Log Pointwise Predictive Density under Leave One Out (ELPD-LOO) metric for time series. Formally, assume observations $y_1, \ldots, y_n$ are independent given parameters θ. Let $p(\theta)$ be a prior distribution on the model parameters that forms a posterior distribution $p(\theta \mid y)$ and posterior predictive $p(\tilde{y} \mid y) = \int p(\tilde{y} \mid \theta) p(\theta \mid y)\, d\theta$. We measure predictive accuracy using the expected log pointwise predictive density:

$$\text{ELPD} - \text{LOO} = \sum_{i=1}^{n} \int p_t(\tilde{y}_i)\, \log p(\tilde{y}_i \mid y)\, d\tilde{y}_i \tag{1}$$

where $p_t(\tilde{y})$ denotes the true data-generating distribution that we use LOO to approximate. ELPD measures predictive accuracy on a log-probability scale, with higher values indicating better fit. While unbounded below, its absolute value is less important than differences between models, which quantify relative predictive performance. Due to the computational cost of running ELPD-LOO over the full time series, we apply a sliding-window sampling procedure to create a representative subsample (25%) of the signal. If a sample's ELPD is more than 4 standard deviations from the mean for its agent backbone and method, we flag it as an outlier and re-fit it on the full time series, dividing the ELPD by 4 to normalize (since it is the sum of point-wise calculations). This affects 10 of 630 samples (9 BoxLM, 1 PyVision). All time-series results in the paper use the re-fit values in order to minimize the impact of sub-sampling at 25%.

We consider additional metrics for both distribution fitting and time-series modeling in Appendix E.5. Results under alternative metrics do not change the rank of the best performing agentic systems.

4.3 Dataset Setup and Statistics

Let $\bar{D} = \{D_1, \ldots, D_n\}$ be a collection of datasets where each D_i represents a set of $n \in [600, 1500]$ points randomly sampled from a ground-truth probability distribution or time-series signal D_{True_i}, whose parameters are drawn uniformly from fixed ranges (see Appendix C). For each domain and difficulty split we generate 50 (**Easy**/**Astro**) or 100 (**Hard**) such datasets, evaluated against the known ground-truth model. In total, our benchmark consists of 200 discrete probability distributions and 200 time series.

5 Experimental Setup

5.1 VESTA Implementation Details

We use the following models as backbones for our VESTA: GPT-5.4-mini, Claude Sonnet 4.6, and Kimi K2.5. For GPT-5.4-mini the reasoning effort is set to "low" and for Claude Sonnet 4.6 we select the minimal number of reasoning tokens (1024) due to the iterative nature of our task and exploding token lengths. For all experiments run with VESTA we set the maximum number of iterations $N = 5$, number of proposals per step $p = 3$, and we select R to be the Akaike Information Criterion (AIC) as

it balances model complexity with fit and does not need a ground truth distribution. For the initial plot, we use a histogram for distributions and simple line plot for time series (See Figure 2).

Expert Visual Tools In addition to dynamically generating tools, we can also equip VESTA with tools designed by an expert statistician as an *oracle* experiment. The expert toolkit remains unaltered during the course of exploration and consists of general-purpose functions that target key questions expert statisticians use when fitting complex mixture distributions or noisy time-series. For distribution fitting, the static toolkit includes tools for visualizing QQ plots, tail transform diagnostics, probability plots, and running a Gaussian-Mixture-Model based segmentation with moment estimation, enabling agents to systematically diagnose location, scale, tail, and mixture structure miscalculations. For time series, the expert toolkit includes tools for comparing model fit against actuals, diagnosing residuals, and testing temporal independence via autocorrelation plots and Ljung-Box statistics, together enabling agents to detect underfitting, missed periodicity, and unmodeled dynamics. A complete description of tools is available in Appendix F.

5.2 Baselines

We detail exact prompts of all baselines in Appendix I. For fair comparison with VESTA, we set the maximum number of iterations $N = 5$ for the BoxLM and PyVision baselines. For Claude Code, Codex, and OpenCode, we provide a computational budget that matches the "worst case" of VESTA.

BoxLM. BoxLM [27] is inspired by the principle of Box's Loop where a language model proposes statistical models using PyMC code and then critiques the probabilistic program over N rounds of iteration. The LLM proposes M models per round and scores each of them using ELPD-LOO, selecting the top p as candidates for the next round. The critic LM then synthesizes the fitted models, their scores, and posterior predictive summary statistics into a natural language hypothesis in order to update the model proposals in the next round of feedback. We minimally adapt BoxLM prompts to better reflect our task goals.

PyVision. PyVision [55] is a multi-turn framework designed to enhance multimodal reasoning by allowing VLMs to generate and execute Python code during inference. A VLM receives an input, generates Python code, and executes it in an isolated runtime, with the resulting output fed back into the VLM's context to refine reasoning over multiple turns. The iteration continues until the VLM decides to provide a final answer. We adapt PyVision to propose and fit PyMC models by providing the same intent and PyMC coding guidelines that Visual Exploration Agent receives, keeping the multi-turn execution loop intact.

Agentic Coding Harnesses. To test whether a general-purpose coding agent can match VESTA's structured exploration, we run each backbone model in its native coding harness. Namely, we use Claude Code for Claude Sonnet 4.6, Codex for GPT-5.4-mini, and OpenCode for Kimi-K2.5. A new session is initialized for each dataset, with code executed in a sandboxed directory containing the data, the initial plot, shell access, standard file read/write/search tools, and a Python environment with PyMC, NumPy, pandas, SciPy, and matplotlib. For each session, we set a computational budget matching VESTA, i.e. stopping the run after proposal and fit of 18 PyMC models or making 54 LLM calls, whichever comes first, and then selecting the best-performing model.

6 Results and Analysis on DAWN

Distribution Fitting Results Figure 4 reports the Jensen-Shannon Divergence between the ground truth distribution and the probability density function induced by agent-written PyMC code. VESTA equipped with tool use consistently outperforms both the PyVision and BoxLM baselines on the distribution fitting tasks in DAWN. Coding harnesses (Codex, Claude Code, OpenCode) outperform VESTA on the **Easy** and **Astro** splits, whereas VESTA performs better on the **Hard** mixtures. Inspection of agent traces suggests that coding agents use established numerical fitting libraries to explore popular distributions before submitting PyMC models. Such workflows are effective for single-family problems. Mixture fitting, however, introduces additional choices about component families, weights, and initialization, making effective diagnosis of the current model's discrepancies more consequential. VESTA's structured, model-conditioned feedback is particularly useful for such compositional modeling.

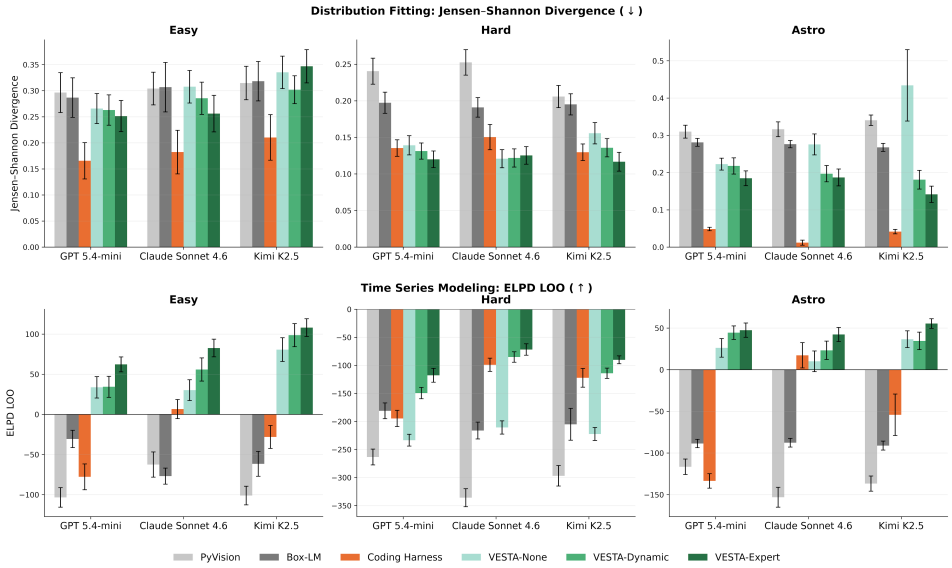


Figure 4: **[Top]** Average Jensen-Shannon divergence (↓ better) between the ground-truth distribution and the probability density function of the proposed PyMC model on the Distribution Fitting task of DAWN. **[Bottom]** Average ELPD-LOO (↑ better) for the Time Series Modeling task of DAWN, computed via leave-one-out cross-validation. Error bars denote ±1 standard error of the mean.

When provided with an expert toolkit, VESTA achieves the strongest overall performance, establishing a reference configuration for what tool discovery pipelines can attain. Crucially, VESTA with dynamic tool generation approaches this reference without access to any expert-written tools, producing distributions that far more closely reflect ground truth structure than the no-tool baseline. This gap demonstrates that VESTA is an effective tool discovery method.

Time Series Results In contrast to Distribution Fitting, VESTA with either dynamic or expert toolkits consistently outperforms PyVision, BoxLM and all coding harnesses, across all three Time Series data splits. VESTA equipped with the expert toolkit has the strongest performance, followed closely by VESTA with the dynamic toolkit. We find that current baselines especially struggle to effectively model time series in DAWN.

6.1 Impact of Tool Persistence and Generation in VESTA

To test the impact of tool accumulation, we sample 100 instances from the **Hard** distribution-fitting split and evenly partition them into 50 tool discovery and 50 test examples. In the tool discovery phase, VESTA fits PyMC models to the 50 tool discovery examples sequentially with tool accumulation enabled. Each run is initialized with all tools generated in prior runs, and the model either selects an existing tool or writes a new one. We find that accumulation converges quickly, with the registry containing only 4 unique tools after 50 runs. We call the registry after the tool discovery phase the *accumulated toolkit*. For all VESTA variants, we use a Claude Sonnet 4.6 backbone and report mean Jensen-Shannon divergence on the same 50 test examples. We evaluate four new variants of VESTA. In VESTA-AccOnly, we initialize with the accumulated toolkit and do not allow further tool generation, while in VESTA-Acc+Gen we also allow new tools to be created on the test set. VESTA-Expert+Gen starts with the expert toolkit but allows new tools to be created on the test set. In VESTA-Ephemeral, we begin with an empty toolkit and allow generation, but no tools are stored. We benchmark these against VESTA-Expert and VESTA-Dynamic on the same test set. Table 1 summarizes each condition and its results.

The results shown in Table 1 indicate that tool generation is a particularly important component of VESTA. In all cases, we see that the ability to generate new tools on unseen examples improves performance. Generation allows VESTA to construct diagnostics targeted at the specific discrepancies between the current model and the data, which we hypothesize is the reason that this improvement is consistent for all starting toolkit initializations. The gain from VESTA-Expert to VESTA-Expert+Gen further suggests that generated tools complement expert-created tools rather than reproduce them

Table 1: Tool persistence ablation on 50 held-out Hard distribution-fitting examples (Claude Sonnet 4.6). Columns indicate the starting toolkit, whether generation is allowed at test time, and whether generated tools persist across test runs. Mean JS divergence, lower is better.

Condition	Starting toolkit	Generation	Persistence	Mean JSD
VESTA-AccOnly	Accumulated toolkit	✗	—	0.123
VESTA-Acc+Gen	Accumulated toolkit	✓	✓	0.115
VESTA-Expert+Gen	Expert toolkit	✓	✓	**0.106**
VESTA-Ephemeral	Empty	✓	✗	0.120
VESTA-Expert	Expert toolkit	✗	—	0.129
VESTA-Dynamic	Empty	✓	✓	0.124

directly. Finally, the small advantage of VESTA-Ephemeral over VESTA-Dynamic is consistent with a growing registry biasing the agent toward reusing existing tools over constructing more targeted ones, although the difference is small and may not be significant at this sample size.

6.2 Analysis of Generated Tools

Case Study Figure 5 shows an example of a VESTA-written tool designed to analyze a right-tailed distribution. The tool generated a 6-panel visualization. The original histogram of the input distribution is in **Panel 4**.

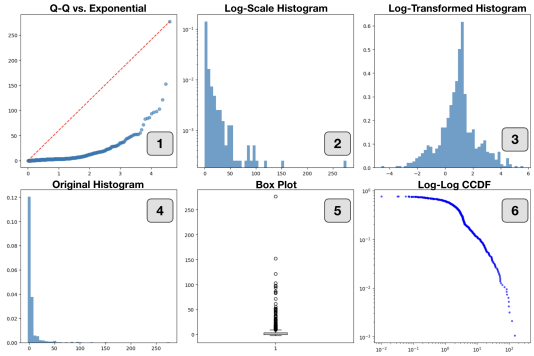


Figure 5: Example of a VESTA-generated tool. This multi-panel output is fed back into VESTA to generate better hypotheses. Panel titles are enlarged for clarity and panel numbers are added manually.

Each of the remaining subplots are designed to test particular hypotheses about this skewed distribution. **Panel 1** shows a Q-Q plot testing whether the current proposed exponential-distribution-based model is appropriate for this data, shown by whether points fall along the reference line. **Panel 2** plots a histogram of the raw data on a log-scaled x-axis, which compresses the right tail and spreads out the lower values, making it easier to assess the shape of the distribution. **Panel 3** plots a histogram of the log-transformed data, meaning each observation is logged before being binned. If this histogram appears bell-shaped and symmetric then a log-normal family is appropriate for the original data. **Panel 5** is a box plot and **Panel 6** plots the complementary cumulative distribution function (CCDF) of the data in log-log space to see if the data follows a power-law distribution. This multi-panel visualization is a very powerful ingredient for VESTA; one limitation is that VLMs sometimes exhibit a perceptual "blindness" [38, 45] and fail to analyze these complex visualizations, a technical limitation that if addressed, will likely yield further performance gains.

Generated vs. Expert Tools We analyze the tools generated by VESTA and PyVision using Claude Sonnet 4.6, and compare them to the expert-written toolkit. We find that VESTA and PyVision often independently "discover" tools present in the expert toolkit and compose them together to create a single, powerful tool for analysis. In contrast, expert tools are atomic by design, often testing at most two hypotheses. For example, a single tool generated by VESTA composes an average of 3.5 tools present in the expert toolkit, compared to 1.9 tools for PyVision.

Table 2 quantifies this analysis at scale. We calculate the percent of times a generated tool contains a sub-function present in the expert toolkit. First, we find that both methods discover each function type in the toolkit at least once. VESTA recovers a much larger percentage of expert tools than PyVision.

Second, we find that generated tools depend on the unique specifications of the task. For distribution fitting, the **Astro** split contains Initial Mass Functions (IMFs) which are exclusively power law or mixtures of power law distributions. In order to visually distinguish these distributions, applying a

Table 2: Expert tool coverage (%) across difficulty levels for VESTA and PyVision using Claude Sonnet 4.6. Each value is the percentage of generated tools that implement one of the expert tools.

	Easy		Hard		Astro	
Expert Tool	**VESTA**	**PyVision**	**VESTA**	**PyVision**	**VESTA**	**PyVision**
Distributions						
Calculate Moments	**98.2**	72.7	**94.6**	74.4	**98.4**	88.0
Histogram	**98.2**	14.7	**99.2**	22.8	**98.4**	2.7
Tails Transforms	**55.4**	44.7	**56.2**	36.7	**91.8**	65.3
QQ Plot	**60.7**	4.0	**36.2**	1.1	**23.0**	0.7
Segment & CM.	**23.2**	9.3	**56.2**	38.3	**14.8**	0.0
Probability Plot	26.8	**30.7**	**26.9**	22.8	**39.3**	32.0
Time Series						
Residuals ACF Plot	**93.8**	49.3	**84.1**	41.5	**97.1**	50.7
Fit vs. Actuals	**85.2**	57.3	**77.9**	52.4	**67.6**	56.0
Instantaneous Freq. Plot	**88.9**	61.3	**59.3**	42.4	**82.4**	58.0
Fit vs. Actuals w/ Res. Dist.	**76.5**	7.3	**72.4**	4.2	**58.8**	0.0
Residuals ACF Score	14.8	**32.0**	9.7	**18.2**	13.2	**36.7**

log or log-log transform is required. Accordingly, for VESTA the percent of generated tools using a "tail transform" function increases from 55.4% on the Easy split to 91.8% on the Astro split.

VESTA Created Tools Beyond the Expert Toolkit Not only does VESTA discover expert-written tools, but it creates tools that further strengthen analysis. Table 3 shows the percentage of VESTA-generated tools that contain a function not present in the expert toolkit. We find that nearly all generated tools contain some form of "diagnostic check" where the tool fits one or more candidate distributions against the input data to assess which distribution best models the data. This is often accompanied by a formal ranking by calculating different "information criteria" metrics such as AIC or BIC. We find the next most common functions (mean-excess and Hill estimator) are predominantly called on the Astro split and are designed to diagnose heavy-tailed behavior. The mean-excess plot exploits linearity of $\mathbb{E}[X - u \mid X > u]$ in the threshold u to identify a Generalized-Pareto-like tail and locate where it begins, while the Hill estimator provides a direct estimate of the tail index α from the largest order statistics. Their focus on Astro suggests that VESTA correctly identifies Initial Mass Functions as tail-dominated distributions and tailors its tools to target these features. The remaining tools (Box–Cox, D'Agostino's normality test, and Shapiro–Wilk) are all related to Gaussian distributions, either testing for normality or transforming data to be normally distributed. A detailed description of each tool, as well as an analysis of novel time-series tools is presented in Appendix G.

Table 3: For the distribution fitting task, the percentage of VESTA-generated tools that invoke each function as a subroutine. Only functions not present in the expert toolkit are shown.

Function	Easy	Hard	Astro
Diagnostic Fit Checks	**100.0**	96.9	**100.0**
Info. Criteria	**82.1**	73.8	65.6
Mean Excess Plot	5.4	7.7	**36.1**
Hill Estimator	0.0	1.5	**6.6**
Box–Cox	**1.8**	0.0	0.0
D'Agostino Normaltest	0.0	**0.8**	0.0
Shapiro–Wilk	0.0	**0.8**	0.0

7 Conclusion

We introduce VESTA, a pipeline that uses VLMs to propose statistical models and test hypotheses by way of visual diagnostic tools, producing models more accurately reflecting the underlying data structure. To support this evaluation, we introduce DAWN, a benchmark of distribution fitting and time series modeling tasks of increasing difficulty, including real-world astronomy problems (Astro). Our evaluation finds that VESTA's dynamic tool generation both independently recovers expert tools and composes them into sophisticated diagnostics to test multiple hypotheses simultaneously. While expert tools achieve the strongest performance, VESTA outperforms existing baselines on complex distribution fitting and time-series modeling tasks, demonstrating that autonomous discovery of visual diagnostic tools can lead to fitting higher-quality statistical models.

Acknowledgments

This work was supported by the NSF under Cooperative Agreement 2421782 and the Simons Foundation grant MPS-AI-00010515 awarded to the NSF-Simons AI Institute for Cosmic Origins - CosmicAI, https://www.cosmicai.org/. It was also partially supported by NSF grants IIS-2145280, IIS-2433071, and CCF-2403036. We would also like to thank Stéphanie Juneau from CosmicAI / NOIRLab for her help and feedback on this work.

References

[1] Oriol Abril-Pla, Virgile Andreani, Colin Carroll, Larry Dong, Christopher J Fonnesbeck, Maxim Kochurov, Ravin Kumar, Junpeng Lao, Christian C Luhmann, Osvaldo A Martin, et al. Pymc: a modern, and comprehensive probabilistic programming framework in python. *PeerJ Computer Science*, 9:e1516, 2023.

[2] Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems, 2026. URL https://arxiv.org/abs/2603.02766.

[3] Ali Aroudi, Hadi Veisi, Hossein Sameti, and Zahra Mafakheri. Speech signal modeling using multivariate distributions. *EURASIP Journal on Audio Speech and Music Processing*, 2015: 1–14, 12 2015. doi: 10.1186/s13636-015-0078-1.

[4] Nate Bastian, Kevin R. Covey, and Michael R. Meyer. A universal stellar initial mass function? a critical look at variations. *Annual Review of Astronomy and Astrophysics*, 48(Volume 48, 2010):339–389, 2010. ISSN 1545-4282. doi: https://doi.org/10.1146/annurev-astro-082708-101642. URL https://www.annualreviews.org/content/journals/10.1146/annurev-astro-082708-101642.

[5] Josh Bongard and Hod Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences of the United States of America*, 104(24): 9943–9948, Jun 2007. doi: 10.1073/pnas.0609476104.

[6] Jure Brence, Ljupco Todorovski, and Saso Dzeroski. Probabilistic grammars for equation discovery. *CoRR*, abs/2012.00428, 2020. URL https://arxiv.org/abs/2012.00428.

[7] Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers, 2024. URL https://arxiv.org/abs/2305.17126.

[8] Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, Alex Dimakis, and Ion Stoica. Adaevolve: Adaptive llm driven zeroth-order optimization, 2026. URL https://arxiv.org/abs/2602.20133.

[9] Gilles Chabrier. Galactic stellar and substellar initial mass function. *Publications of the Astronomical Society of the Pacific*, 115(809):763–795, July 2003. ISSN 1538-3873. doi: 10.1086/376392. URL http://dx.doi.org/10.1086/376392.

[10] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. Mle-bench: Evaluating machine learning agents on machine learning engineering, 2025. URL https://arxiv.org/abs/2410.07095.

[11] Gangda Deng, Zhaoling Chen, Zhongming Yu, Haoyang Fan, Yuhong Liu, Yuxin Yang, Dhruv Parikh, Rajgopal Kannan, Le Cong, Mengdi Wang, Qian Zhang, Viktor Prasanna, Xiangru Tang, and Xingyao Wang. Evoclaw: Evaluating ai agents on continuous software evolution, 2026. URL https://arxiv.org/abs/2603.13428.

[12] David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search, 2013. URL https://arxiv.org/abs/1302.4922.

[13] Alex Egg, Martin Iglesias Goyanes, Friso Kingma, Andreu Mora, Leandro von Werra, and Thomas Wolf. Dabstep: Data agent benchmark for multi-step reasoning, 2025. URL https://arxiv.org/abs/2506.23719.

[14] Yasaman Ensafi, Saman Hassanzadeh Amin, Guoqing Zhang, and Bharat Shah. Time-series forecasting of seasonal items sales using machine learning – a comparative analysis. *International Journal of Information Management Data Insights*, 2(1):100058, 2022. ISSN 2667-0968. doi: 10.1016/j.jjimei.2022.100058. URL https://www.sciencedirect.com/science/article/pii/S2667096822000027.

[15] Kanishk Gandhi, Michael Y. Li, Lyle Goodyear, Agam Bhatia, Louise Li, Aditi Bhaskar, Mohammed Zaman, and Noah D. Goodman. Boxinggym: Benchmarking progress in automated experimental design and model discovery, 2025. URL https://arxiv.org/abs/2501.01540.

[16] Nate Gruver, Marc Finzi, Shikai Qiu, and Andrew Gordon Wilson. Large language models are zero-shot time series forecasters, 2024. URL https://arxiv.org/abs/2310.07820.

[17] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training, 2022. URL https://arxiv.org/abs/2211.11559.

[18] Jack Hong, Chenxiao Zhao, ChengLin Zhu, Weiheng Lu, Guohai Xu, and Xing Yu. Deepeyesv2: Toward agentic multimodal model, 2026. URL https://arxiv.org/abs/2511.05271.

[19] Xinhai Hou, Shaoyuan Xu, Manan Biyani, Moyan Li, Jia Liu, Todd C. Hollon, and Bryan Wang. Codev: Code with images for faithful visual reasoning via tool-aware policy optimization, 2026. URL https://arxiv.org/abs/2511.19661.

[20] Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models, 2024. URL https://arxiv.org/abs/2406.09403.

[21] Xu Huang, Weiwen Liu, Xingshan Zeng, Yuefeng Huang, Xinlong Hao, Yuxian Wang, Yirong Zeng, Chuhan Wu, Yasheng Wang, Ruiming Tang, and Defu Lian. Toolace-dev: Self-improving tool learning via decomposition and evolution, 2025. URL https://arxiv.org/abs/2505.07512.

[22] Wenlong Ji, Weizhe Yuan, Emily Getzen, Kyunghyun Cho, Michael I. Jordan, Song Mei, Jason E Weston, Weijie J. Su, Jing Xu, and Linjun Zhang. An overview of large language models for statisticians, 2025. URL https://arxiv.org/abs/2502.17814.

[23] Sebastian Antony Joseph, Syed Murtaza Husain, Stella SR Offner, StÃŠphanie Juneau, Paul Torrey, Adam S Bolton, Juan P Farias, Niall Gaffney, Greg Durrett, and Junyi Jessy Li. Astrovisbench: A code benchmark for scientific computing and visualization in astronomy. *arXiv preprint arXiv:2505.20538*, 2025.

[24] Lee Jung-Mok, Nam Hyeon-Woo, Moon Ye-Bin, Junhyun Nam, and Tae-Hyun Oh. Automated model discovery via multi-modal & multi-step pipeline, 2025. URL https://arxiv.org/abs/2509.25946.

[25] P. Kroupa. On the variation of the initial mass function. *Monthly Notices of the Royal Astronomical Society*, 322(2):231–246, April 2001. ISSN 1365-2966. doi: 10.1046/j.1365-8711.2001.04022.x. URL http://dx.doi.org/10.1046/j.1365-8711.2001.04022.x.

[26] Hongwei Li, Zhun Wang, Qinrun Dai, Yuzhou Nie, Jinjun Peng, Ruitong Liu, Jingyang Zhang, Kaijie Zhu, Jingxuan He, Lun Wang, Yangruibo Ding, Yueqi Chen, Wenbo Guo, and Dawn Song. Opensage: Self-programming agent generation engine, 2026. URL https://arxiv.org/abs/2602.16891.

[27] Michael Y. Li, Emily B. Fox, and Noah D. Goodman. Automated statistical model discovery with language models, 2024. URL https://arxiv.org/abs/2402.17879.

[28] Michael Y. Li, Vivek Vajipey, Noah D. Goodman, and Emily B. Fox. Critical: Critic automation with language models, 2024. URL https://arxiv.org/abs/2411.06590.

[29] James Robert Lloyd, David Duvenaud, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Automatic construction and natural-language description of nonparametric regression models, 2014. URL https://arxiv.org/abs/1402.4304.

[30] Jiaxuan Lu, Ziyu Kong, Yemin Wang, Rong Fu, Haiyuan Wan, Cheng Yang, Wenjie Lou, Haoran Sun, Lilong Wang, Yankai Jiang, Xiaosong Wang, Xiao Sun, and Dongzhan Zhou. Beyond static tools: Test-time tool evolution for scientific reasoning, 2026. URL https://arxiv.org/abs/2601.07641.

[31] Ross Maller, Sidney Resnick, Soudabeh Shemehsavar, and Muzhi Zhao. Mixture cure model methodology in survival analysis: Some recent results for the one-sample case. *Statistics Surveys*, 18, 01 2024. doi: 10.1214/24-SS147.

[32] NASA Science. Vesta: In depth. https://science.nasa.gov/solar-system/asteroids/4-vesta/, . Accessed: May 2, 2026.

[33] NASA Science. Dawn mission overview. https://science.nasa.gov/mission/dawn/, . Accessed: May 2, 2026.

[34] Jingchao Ni, Ziming Zhao, ChengAo Shen, Hanghang Tong, Dongjin Song, Wei Cheng, Dongsheng Luo, and Haifeng Chen. Harnessing vision models for time series analysis: A survey, 2025. URL https://arxiv.org/abs/2502.08869.

[35] S. S. R. Offner, P. C. Clark, P. Hennebelle, N. Bastian, M. R. Bate, P. F. Hopkins, E. Moreaux, and A. P. Whitworth. *The Origin and Universality of the Stellar Initial Mass Function*. University of Arizona Press, 2014. ISBN 9780816531240. doi: 10.2458/azu_uapress_9780816531240-ch003. URL http://dx.doi.org/10.2458/azu_uapress_9780816531240-ch003.

[36] Cheng Qian, Chi Han, Yi R. Fung, Yujia Qin, Zhiyuan Liu, and Heng Ji. Creator: Tool creation for disentangling abstract and concrete reasoning of large language models, 2024. URL https://arxiv.org/abs/2305.14318.

[37] James Requeima, John Bronskill, Dami Choi, Richard E. Turner, and David Duvenaud. Llm processes: Numerical predictive distributions conditioned on natural language, 2024. URL https://arxiv.org/abs/2405.12856.

[38] William Rudman, Michal Golovanevsky, Amir Bar, Vedant Palit, Yann LeCun, Carsten Eickhoff, and Ritambhara Singh. Forgotten polygons: Multimodal large language models are shape-blind. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar, editors, *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11983–11998, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.620. URL https://aclanthology.org/2025.findings-acl.620/.

[39] Edwin E. Salpeter. The Luminosity Function and Stellar Evolution. *apj*, 121:161, January 1955. doi: 10.1086/145971.

[40] Chenglei Si, Zitong Yang, Yejin Choi, Emmanuel Candès, Diyi Yang, and Tatsunori Hashimoto. Towards execution-grounded automated ai research, 2026. URL https://arxiv.org/abs/2601.14525.

[41] Yifan Song, Weimin Xiong, Dawei Zhu, Wenhao Wu, Han Qian, Mingbo Song, Hailiang Huang, Cheng Li, Ke Wang, Rong Yao, Ye Tian, and Sujian Li. Restgpt: Connecting large language models with real-world restful apis, 2023. URL https://arxiv.org/abs/2306.06624.

[42] Maojun Sun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan, and Jian Huang. A survey on large language model-based agents for statistics and data science. *The American Statistician*, page 1–14, October 2025. ISSN 1537-2731. doi: 10.1080/00031305.2025.2561140. URL http://dx.doi.org/10.1080/00031305.2025.2561140.

[43] Zeyi Sun, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiaoyi Dong, Tong Wu, Dahua Lin, and Jiaqi Wang. Seagent: Self-evolving computer use agent with autonomous learning from experience, 2025. URL https://arxiv.org/abs/2508.04700.

[44] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning, 2023. URL https://arxiv.org/abs/2303.08128.

[45] Shengbang Tong, Zhuang Liu, Yuexiang Zhai, Yi Ma, Yann LeCun, and Saining Xie. Eyes wide shut? exploring the visual shortcomings of multimodal llms, 2024. URL https://arxiv.org/abs/2401.06209.

[46] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL https://arxiv.org/abs/2305.16291.

[47] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey, 2023. URL https://arxiv.org/abs/2202.07125.

[48] Chenfei Wu, Shengming Yin, Weizhen Qi, Xiaodong Wang, Zecheng Tang, and Nan Duan. Visual chatgpt: Talking, drawing and editing with visual foundation models, 2023. URL https://arxiv.org/abs/2303.04671.

[49] Georg Wölflein, Dyke Ferber, Daniel Truhn, Ognjen Arandjelović, and Jakob Nikolas Kather. Llm agents making agent tools, 2025. URL https://arxiv.org/abs/2502.11705.

[50] Shilin Yan, Jintao Tong, Hongwei Xue, Xiaojun Tang, Yangyang Wang, Kunyu Shi, Guannan Zhang, Ruixuan Li, and Yixiong Zou. Act wisely: Cultivating meta-cognitive tool use in agentic multimodal models, 2026. URL https://arxiv.org/abs/2604.08545.

[51] Xinlei Yu, Chengming Xu, Guibin Zhang, Zhangquan Chen, Yudong Zhang, Yongbo He, Peng-Tao Jiang, Jiangning Zhang, Xiaobin Hu, and Shuicheng Yan. Vismem: Latent vision memory unlocks potential of vision-language models, 2026. URL https://arxiv.org/abs/2511.11007.

[52] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. A transformer-based framework for multivariate time series representation learning, 2020. URL https://arxiv.org/abs/2010.02803.

[53] Yifan Zhang, Liang Hu, Haofeng Sun, Peiyu Wang, Yichen Wei, Shukang Yin, Jiangbo Pei, Wei Shen, Peng Xia, Yi Peng, Tianyidan Xie, Eric Li, Yang Liu, Xuchen Song, and Yahui Zhou. Skywork-r1v4: Toward agentic multimodal intelligence through interleaved thinking with images and deepresearch, 2025. URL https://arxiv.org/abs/2512.02395.

[54] Zhehao Zhang, Ryan Rossi, Tong Yu, Franck Dernoncourt, Ruiyi Zhang, Jiuxiang Gu, Sungchul Kim, Xiang Chen, Zichao Wang, and Nedim Lipka. Vipact: Visual-perception enhancement via specialized vlm agent collaboration and tool-use, 2025. URL https://arxiv.org/abs/2410.16400.

[55] Shitian Zhao, Haoquan Zhang, Shaoheng Lin, Ming Li, Qilong Wu, Kaipeng Zhang, and Chen Wei. Pyvision: Agentic vision with dynamic tooling, 2025. URL https://arxiv.org/abs/2507.07998.

[56] Siru Zhong, Weilin Ruan, Ming Jin, Huan Li, Qingsong Wen, and Yuxuan Liang. Time-vlm: Exploring multimodal vision-language models for augmented time series forecasting, 2025. URL https://arxiv.org/abs/2502.04395.

[57] Qiji Zhou, Ruochen Zhou, Zike Hu, Panzhong Lu, Siyang Gao, and Yue Zhang. Image-of-thought prompting for visual reasoning refinement in multimodal large language models, 2024. URL https://arxiv.org/abs/2405.13872.

[58] Zetong Zhou, Dongping Chen, Zixian Ma, Zhihan Hu, Mingyang Fu, Sinan Wang, Yao Wan, Zhou Zhao, and Ranjay Krishna. Reinforced visual perception with tools, 2025. URL https://arxiv.org/abs/2509.01656.

[59] Yizhang Zhu, Shiyin Du, Boyan Li, Yuyu Luo, and Nan Tang. Are large language models good statisticians?, 2024. URL https://arxiv.org/abs/2406.07815.

A Author contributions

William Rudman, Abhishek Divekar, and Kanishk Jain contributed equally, with complementary responsibilities.

William Rudman led the overall research direction, study design, and scientific framing. He developed the initial modeling framework, led distribution-fitting benchmark development, evaluation, and analysis, and coordinated research across modeling domains and with domain experts. He led manuscript preparation and the synthesis of scientific findings.

Abhishek Divekar led the development of VESTA's central dynamic visual tool-creation methodology. He was the principal implementer of the unified framework and co-led computational experimentation and validation across distribution-fitting and time-series tasks, contributing to study design, analysis, and manuscript revision. His work on this paper does not relate to his affiliation with Amazon.

Kanishk Jain led the development of the time-series forecasting methodology and its dedicated modeling framework, including the workflow, dataset development, domain-specific tools, and performance evaluation. He worked with William Rudman and Abhishek Divekar to unify the time-series and distribution-fitting frameworks into a common, production-ready design for final experiments. He generated and analyzed time-series results and contributed to manuscript preparation.

Sebastian Joseph contributed to research discussions and exploratory baseline development.

Stella S. R. Offner provided astrophysical expertise and guidance on scientific applications. **Matthew Lease** and **Kyle Mahowald** provided research guidance and feedback on the project's development and scientific framing.

Greg Durrett and **Junyi Jessy Li** contributed equally to advising and supervision. They guided the formulation of research questions, methodological development, and experimental design. They helped position the work within the relevant literature, interpret findings, and refine the paper's scientific claims, and contributed to manuscript revision.

B VESTA Details

C DAWN Details.

C.1 Distribution Fitting

For each D_{True_i} in the distribution fitting domain, ground-truth parameters are drawn independently and uniformly from predefined ranges, with the family (or mixture of families) depending on the difficulty split. Nine base families are used throughout: Pareto, Gaussian, Log-normal, Student-t, Exponential, Uniform, Weibull, Cauchy, and Laplace.

Easy split. A single distribution family is drawn uniformly at random from the eight families above. Parameters are then sampled as follows. For the *Gaussian*, mean $\mu \sim \mathcal{U}(-5, 5)$ and standard deviation $\sigma \sim \mathcal{U}(0.5, 3)$. For the *Log-normal*, log-mean $\mu \sim \mathcal{U}(0, 2)$ and log-width $\sigma \sim \mathcal{U}(0.3, 1.5)$. For the *Student-t*, location $\mu \sim \mathcal{U}(-5, 5)$, scale $\sigma \sim \mathcal{U}(0.5, 3)$, and degrees of freedom $\nu \sim \mathcal{U}(2.5, 10)$. For the *Exponential*, location $\ell \sim \mathcal{U}(0, 2)$ and scale $\lambda \sim \mathcal{U}(0.5, 3)$. For the *Uniform*, mean $\mu \sim \mathcal{U}(-5, 5)$ and standard deviation $\sigma \sim \mathcal{U}(0.5, 3)$, from which the bounds are derived as $[\mu - \sigma\sqrt{3},\ \mu + \sigma\sqrt{3}]$. For the *Weibull*, location $\ell \sim \mathcal{U}(0, 2)$, shape $\alpha \sim \mathcal{U}(1, 3)$, and scale $\lambda \sim \mathcal{U}(0.5, 2)$. For the *Cauchy*, location $\alpha \sim \mathcal{U}(-5, 5)$ and scale $\beta \sim \mathcal{U}(0.5, 2)$. For the *Laplace*, location $\mu \sim \mathcal{U}(-5, 5)$ and diversity $b \sim \mathcal{U}(0.5, 2)$. Each dataset contains $n \in [500, 1500]$ points.

Hard split. Two distribution families are drawn independently with replacement from the same eight families. Each component draws its own sample size $n_k \in [250, 750]$ independently, and parameters for each component are sampled using the same per-family ranges described above. The two components are concatenated to form the final dataset of total size $n = n_1 + n_2$.

Astro split. Each ground-truth model is one of five initial mass function (IMF) families: Salpeter, Kroupa, Chabrier, Free-form (wide), and Free-form (tight), with 20 independent realisations generated

Algorithm 2 Visual Exploration Agents (Detailed)

Require: Data D, iteration limit N, proposals per iteration p, metric R, registry $\mathcal{E}$ (initial state: `generate_new_tool` only)
Ensure: $M_{\text{best}}, \theta_{\text{best}}$

$P_0 \leftarrow \text{Plot}(D)$
$\{M_0^{(j)}\}_{j=1}^{p} \leftarrow \textsc{Propose}(P_0)$ ▷ Proposal Agent proposes p diverse models
$\theta_0^{(j)} \leftarrow \textsc{StatGen}(D, M_0^{(j)})\ \forall j$ ▷ Stat-Gen fits all p models
$M_0, \theta_0 \leftarrow \arg\min_j\ R(D, M_0^{(j)}(\theta_0^{(j)}))$ ▷ Select best model under metric R
$M_{\text{best}}, \theta_{\text{best}} \leftarrow M_0, \theta_0$
$s_0 \leftarrow \textsc{Summarize}(M_{\text{best}}, \theta_{\text{best}})$
for $i = 1, \ldots, N$ **do**
  $\varepsilon^* \leftarrow \textsc{ToolManager}(s_{i-1}, D, M_{i-1}, \theta_{i-1}, \mathcal{E})$ ▷ Select or *create* diagnostic tool
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{\varepsilon^*\}$ ▷ Persist newly created tools to registry
  $T_i \leftarrow \varepsilon^*(D, M_{i-1}, \theta_{i-1})$ ▷ Execute tool to produce diagnostic
  desc, $\{M_i^{(j)}\}_{j=1}^{p} \leftarrow \textsc{Critique}(T_i, s_{0:i-1}, D)$ ▷ Propose p revised models
  if desc $=$ `COMPLETE` **then**
    break
  end if
  $\theta_i^{(j)} \leftarrow \textsc{StatGen}(D, M_i^{(j)})\ \forall j$ ▷ Stat-Gen fits all p models
  $M_i, \theta_i \leftarrow \arg\min_j\ R(D, M_i^{(j)}(\theta_i^{(j)}))$ ▷ Select best model under metric R
  $M_{\text{best}}, \theta_{\text{best}} \leftarrow \arg\min_{\theta \in \{\theta_i,\, \theta_{\text{best}}\}} R(D, \mathcal{M}(\theta))$
  $s_i \leftarrow \textsc{Summarize}(M_i, \theta_i, \varepsilon^*)$
end for
return $M_{\text{best}}, \theta_{\text{best}}$

per family. The number of stellar masses drawn per realisation is sampled uniformly from $[600, 1500]$, and all families share $m_{\max} = 120\,M_\odot$. The Salpeter IMF is a single power law $\xi(m) \propto m^{-\alpha}$ with $\alpha \sim \mathcal{U}(1.9, 2.7)$ on $[0.08, 120]\,M_\odot$. The Kroupa IMF is a three-segment broken power law with slopes $p_1 \sim \mathcal{U}(0.1, 0.5)$, $p_2 \sim \mathcal{U}(1.0, 1.6)$, $p_3 \sim \mathcal{U}(1.9, 2.7)$ and fixed break masses $m_1 = 0.08\,M_\odot$, $m_2 = 0.5\,M_\odot$ on $[0.009, 120]\,M_\odot$; continuity constants are enforced at each break. The Chabrier IMF joins a log-normal component ($m_{\text{char}} \sim \mathcal{U}(0.15, 0.35)$, $\sigma \sim \mathcal{U}(0.45, 0.70)$) below $1\,M_\odot$ to a power law ($\alpha \sim \mathcal{U}(1.9, 2.7)$) above $1\,M_\odot$, matched continuously at the junction. The Free-form IMF is an 8-parameter broken power law with four slopes and three interior break masses; the wide variant uses broad exploratory ranges ($a_0 \sim \mathcal{U}(-0.5, 1.0)$, $a_1 \sim \mathcal{U}(0.5, 2.0)$, $a_2 \sim \mathcal{U}(1.5, 3.0)$, $a_3 \sim \mathcal{U}(1.8, 3.5)$; breaks $b_1 \sim \mathcal{U}(0.06, 0.12)$, $b_2 \sim \mathcal{U}(0.30, 0.80)$, $b_3 \sim \mathcal{U}(0.80, 2.00)$; $m_{\min} \sim \mathcal{U}(0.007, 0.040)$), while the tight variant uses ranges anchored near canonical Kroupa values ($a_0 \sim \mathcal{U}(0.1, 0.5)$, $a_1 \sim \mathcal{U}(1.0, 1.6)$, $a_2, a_3 \sim \mathcal{U}(1.9, 2.7)$; breaks $b_1 \sim \mathcal{U}(0.07, 0.09)$, $b_2 \sim \mathcal{U}(0.40, 0.60)$, $b_3 \sim \mathcal{U}(0.80, 1.20)$; $m_{\min} \sim \mathcal{U}(0.007, 0.012)$). Break masses are sorted to enforce $b_1 < b_2 < b_3$, and continuity constants are applied at each break.

C.2 Time Series

For each D_{True_i} in the time-series domain, ground-truth signal parameters are drawn independently and uniformly from predefined ranges, with the specific ranges and signal family depending on the difficulty split.

Easy split. Each ground-truth signal follows a sinusoidal trend model,

$$y(t) = s \cdot \tilde{t} + A \sin(2\pi f \tilde{t}) + \varepsilon(t), \tag{2}$$

where $\tilde{t} \in [0, 1]$ is normalized time. Parameters are sampled as slope $s \sim \mathcal{U}(-2.5, 2.5)$, amplitude $A \sim \mathcal{U}(0.2, 2.0)$, frequency $f \sim \mathcal{U}(1, 12)$ cycles per window, and noise fraction $\sigma/A \sim \mathcal{U}(0.05, 0.15)$ with $\varepsilon(t) \sim \mathcal{N}(0, \sigma^2)$. Approximately 16% of series are further constrained to $s = 0$ to ensure flat-trend instances are represented. All series are mean-centred after generation.

Hard split. A two-level sampling scheme is used. First, a signal category is drawn—*Complex* with probability 0.4 or *Rhythmic* with probability 0.6. Within *Complex*, a type is drawn uniformly from: ARMA (an AR(1) random walk), Heteroscedastic (an expanding-variance process with additive seasonal component), and Logistic (an S-curve trend). Within *Rhythmic*, a type is drawn uniformly from: synthetic ECG (a Gaussian P-QRS-T complex template) and Square wave, with period sampled from $\{40, 60, 80\}$ time steps; small Gaussian noise $\varepsilon(t) \sim \mathcal{N}(0, \sigma^2)$ is added to all rhythmic signals. Pure sine-wave instances are generated but excluded from the final dataset to prevent overlap with the easy split.

Astro split (gravitational wave chirps). Each ground-truth signal follows a linearly swept-frequency (chirp) model,

$$y(t) = A(t)\sin\big(2\pi\big(f_0\,t + \tfrac{1}{2}\beta\,t^2\big)\big) + \varepsilon(t), \tag{3}$$

giving instantaneous frequency $f(t) = f_0 + \beta t$. Parameters are sampled as $f_0 \sim \mathcal{U}(0.3,\ 2.5)$, $\beta \sim \mathcal{U}(0.20,\ 1.00)$, $A \sim \mathcal{U}(0.7,\ 1.8)$, and $\sigma \sim \mathcal{U}(0.07,\ 0.15)$. For 40 of the 50 series, amplitude is constant, $A(t) = A$; for the remaining 10, an exponential decay envelope $A(t) = A\,e^{-t/\tau}$ with $\tau \sim \mathcal{U}(2.0,\ 5.0)$ is applied to model inspiral ringdown behaviour.

All time series consist of $N = 600$ equally spaced observations on a daily grid.

D VESTA Tool Generation Variants

To evaluate whether tools can be effectively accumulated through sequential interaction, we sampled 100 instances from the **Hard** distribution fitting split, partitioning them equally into 50 train and 50 test examples. We evaluate three conditions with Claude Sonnet 4.6. (1) Accumulate-Only: During a train phase, the model processes training examples sequentially with tool accumulation enabled, initializing each run with the tools generated in all prior runs and selecting either an existing tool or generating a new one. After 50 sequential runs, this process converges to just 4 unique tools. During the test phase, the model is restricted to selecting from these accumulated tools with no new generation permitted and no interaction between runs. (2) Generate-Only: Identical to Accumulate-Only, but the test phase additionally permits the model to generate new tools, which are not shared across test runs. (3) Expert Toolkit + Generation: No train phase is used. Instead, the model is initialized with the default expert toolkit on all 50 test examples while also retaining the ability to generate new tools, again without cross-run sharing. This design isolates three factors: the value of accumulated tools alone, the benefit of combining accumulated tools with on-the-fly generation, and the contribution of generation when paired with a fixed expert toolkit.

Figure 6 presents JS divergence scores across the three toolkit conditions on the Hard distribution fitting split. Overall, performance differences across conditions are modest, with all three variants achieving mean JS divergence between 0.106 and 0.124, and overlapping standard errors indicating no statistically significant separation. Nevertheless, a consistent ordering emerges: Expert+Generate achieves the lowest divergence (0.106), followed by Generate-Only (0.115), with Accumulate-Only performing worst (0.124). Two conclusions follow from this pattern. First, tool accumulation alone provides no measurable benefit. Restricting the model to the four auto-accumulated tools at test time yields the weakest performance, suggesting that sequential accumulation over 50 training examples converges to a toolkit that is too narrow to generalize across the full diversity of dataset. Second, on-the-fly tool generation is the primary performance driver. Both conditions that permit test-time generation outperform Accumulate-Only, regardless of whether the base toolkit is accumulated or expert-designed. The marginal advantage of Expert+Generate over Generate-Only further suggests

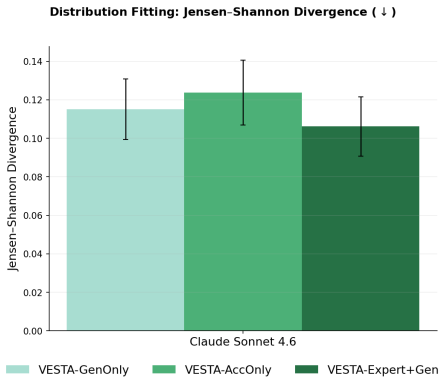


Figure 6: Jensen–Shannon divergence on the **Hard** distribution fitting split (lower is better) comparing three accumulated toolkit conditions with Claude Sonnet 4.6.

that a well-curated expert toolkit provides a stronger generative prior than one derived through automatic accumulation. Taken together, these results indicate that the current accumulation strategy does not yet yield tools competitive with expert-designed ones, and that allowing the model to generate new tools at test time remains essential for robust distribution fitting performance.

E Additional Results on VESTA

E.1 Statistical Significance of the Results

We pair-wise test, for each combination of LLM, dataset split, baseline, and VESTA variant, whether VESTA significantly outperforms the baseline. Both evaluation metrics are treated as independent statistical questions, as: they probe different tasks (distribution fitting vs. time-series modelling), have different data structures, and warrant different tests. Multiple-comparison corrections are applied within each metric only. We compare to BoxLM and PyVision, our primary baseline approaches.

Distribution fitting (JS divergence). Per-problem JS values are available for every method on the same set of problems, so we use a one-sided paired Wilcoxon signed-rank test across problems with $H_1 : \mathrm{JS}_{Vesta} < \mathrm{JS}_{\text{baseline}}$.

Time-series modeling (ELPD LOO). ArviZ's `loo` returns a point estimate $\widehat{\mathrm{ELPD}}$ and its standard error $\widehat{\mathrm{SE}}$ for each fit. We use a one-sided z-test on $\Delta\mathrm{ELPD} = \mathrm{ELPD}_{Vesta} - \mathrm{ELPD}_{\text{baseline}}$ with $\mathrm{SE}_\Delta = \sqrt{\widehat{\mathrm{SE}}_V^{\,2} + \widehat{\mathrm{SE}}_B^{\,2}}$. This SE_Δ is *conservative*: `loo_compare` computes the SE from per-observation ELPD differences, which is strictly smaller whenever the two ELPDs are positively correlated. Our reported p-values are therefore upper bounds on the true p-values.

Multiple comparisons. Within each metric we conduct 3 splits $\times$ 3 LLMs $\times$ 3 VESTA variants $\times$ 2 baselines $= 54$ tests and apply Holm–Bonferroni correction. We use $\alpha = 0.05$ throughout.

E.2 Summary: aggregate win rates

Table 4 reports, per (variant, baseline, split), how many of the 3 LLMs yield a statistically significant win for VESTA after Holm correction. VESTA's advantage on time-series modelling is robust: every variant beats PyVision in every (LLM, split) cell, and beats Box-LM in 13/18 cells. For distribution fitting, significant wins are concentrated on the **Hard** split, where the absolute JS gap between VESTA and the baselines is largest.

Table 4: Number of LLMs (out of 3) for which each VESTA variant significantly outperforms each baseline at $\alpha = 0.05$ after Holm correction. Tests are one-sided in the direction "VESTA beats baseline" (paired Wilcoxon for JS divergence; z-test on ΔELPD for ELPD LOO). Bold marks cells with ≥ 1 significant win.

	Easy		**Hard**		**Astro**	
	PyVis.	Box-LM	PyVis.	Box-LM	PyVis.	Box-LM
Distribution fitting — JS divergence						
VESTA-None	1/3	0/3	**2/3**	0/3	0/3	0/3
VESTA-Dynamic	1/3	0/3	**2/3**	2/3	1/3	0/3
VESTA-Expert	1/3	1/3	**2/3**	**1/3**	**2/3**	**1/3**
Time-series modelling — ELPD LOO						
VESTA-None	**3/3**	**2/3**	**2/3**	0/3	**3/3**	**3/3**
VESTA-Dynamic	**3/3**	**2/3**	**3/3**	**2/3**	**3/3**	**3/3**
VESTA-Expert	**3/3**	**2/3**	**3/3**	**2/3**	**3/3**	**3/3**

E.3 Detailed per-LLM results

Tables 5 and 6 give the median JS gap and ΔELPD respectively for every individual comparison, along with stars denoting Holm-corrected significance levels.

Table 5: Distribution fitting: median JS gap $\Delta = \text{med}(\text{JS}_{Vesta}) - \text{med}(\text{JS}_{\text{baseline}})$. Negative values favour VESTA. Stars are Holm-corrected p-values from the one-sided paired Wilcoxon test: $^{*}p<0.05$, $^{**}p<0.01$, $^{***}p<0.001$. Significant wins ($p_{\text{Holm}}<0.05$) in bold.

		Easy		Hard		Astro	
LLM	**Method**	PyVis.	Box-LM	PyVis.	Box-LM	PyVis.	Box-LM
GPT-5.4-mini	VESTA-None	$+0.084$	$+0.031$	$\mathbf{-0.144}^{**}$	-0.102	-0.001	-0.009
	VESTA-Dynamic	$+0.026$	-0.027	$\mathbf{-0.105}^{*}$	-0.063	-0.023	-0.031
	VESTA-Expert	-0.047	-0.100	$\mathbf{-0.148}^{***}$	$\mathbf{-0.106}^{**}$	-0.078	-0.086
Claude Sonnet 4.6	VESTA-None	-0.024	-0.114	$\mathbf{-0.137}^{**}$	-0.070	-0.034	-0.022
	VESTA-Dynamic	-0.076	-0.166	$\mathbf{-0.149}^{***}$	-0.082	-0.069	-0.057
	VESTA-Expert	-0.076	-0.166	$\mathbf{-0.136}^{***}$	-0.069	$\mathbf{-0.120}^{*}$	-0.108
Kimi K2.5	VESTA-None	$+0.031$	$+0.005$	-0.030	-0.005	$+0.144$	$+0.279$
	VESTA-Dynamic	-0.010	-0.036	-0.083	-0.058	-0.370	-0.235
	VESTA-Expert	-0.004	-0.030	-0.077	-0.052	-0.126	$+0.009$

Table 6: Time-series modelling: $\Delta\text{ELPD} = \text{ELPD}_{Vesta} - \text{ELPD}_{\text{baseline}}$, in nats (positive favours VESTA). Stars are Holm-corrected p-values from the one-sided z-test: $^{*}p<0.05$, $^{**}p<0.01$, $^{***}p<0.001$. Significant wins in bold.

		Easy		Hard		Astro	
LLM	**Method**	PyVis.	Box-LM	PyVis.	Box-LM	PyVis.	Box-LM
GPT-5.4-mini	VESTA-None	$\mathbf{137}^{***}$	$\mathbf{64.3}^{**}$	30	-39	$\mathbf{143}^{***}$	$\mathbf{115}^{***}$
	VESTA-Dynamic	$\mathbf{138}^{***}$	$\mathbf{65.0}^{***}$	$\mathbf{114}^{***}$	$\mathbf{44.8}^{*}$	$\mathbf{161}^{***}$	$\mathbf{133}^{***}$
	VESTA-Expert	$\mathbf{166}^{***}$	$\mathbf{92.9}^{***}$	$\mathbf{146}^{***}$	$\mathbf{76.5}^{***}$	$\mathbf{164}^{***}$	$\mathbf{136}^{***}$
Claude Sonnet 4.6	VESTA-None	$\mathbf{92.7}^{***}$	591	$\mathbf{125}^{***}$	398	$\mathbf{163}^{***}$	$\mathbf{97.7}^{***}$
	VESTA-Dynamic	$\mathbf{118}^{***}$	617	$\mathbf{251}^{***}$	524	$\mathbf{176}^{***}$	$\mathbf{111}^{***}$
	VESTA-Expert	$\mathbf{145}^{***}$	644	$\mathbf{264}^{***}$	537	$\mathbf{195}^{***}$	$\mathbf{130}^{***}$
Kimi K2.5	VESTA-None	$\mathbf{182}^{***}$	$\mathbf{142}^{***}$	$\mathbf{87.2}^{***}$	14.8	$\mathbf{173}^{***}$	$\mathbf{128}^{***}$
	VESTA-Dynamic	$\mathbf{200}^{***}$	$\mathbf{160}^{***}$	$\mathbf{195}^{***}$	$\mathbf{123}^{**}$	$\mathbf{171}^{***}$	$\mathbf{126}^{***}$
	VESTA-Expert	$\mathbf{209}^{***}$	$\mathbf{170}^{***}$	$\mathbf{219}^{***}$	$\mathbf{147}^{***}$	$\mathbf{192}^{***}$	$\mathbf{147}^{***}$

E.4 Discussion

Time-series modelling. VESTA's advantage on ELPD LOO is essentially universal: 45 of 54 comparisons are significant at $\alpha = 0.05$ after Holm correction, with the great majority at $p<0.001$. The five non-significant comparisons against Box-LM on Claude (Easy and Hard splits) are an artefact of Box-LM's catastrophically diffuse posterior on Claude — the ΔELPD point estimates are massive ($+398$ to $+644$ nats in favour of VESTA), but the standard error on the Box-LM side (~ 360 nats) is large enough to keep these just above the Holm-corrected threshold. Substantively, VESTA clearly wins here; the test is simply unable to certify it given how unstable Box-LM's posterior is.

Distribution fitting. Significant wins on JS divergence are concentrated in the **Hard** split, where the effect size is largest. On **Easy** and **Astro**, point estimates favour VESTA in nearly every cell (Table 5), but per-problem variance is high relative to the typical gap, and the Wilcoxon test cannot certify the difference at $\alpha = 0.05$ given 54 corrections. The Kimi/**Astro** results are particularly noisy: VESTA-Dynamic posts a -0.370 median JS gap vs. PyVision yet $p_{\text{Holm}} = 1$, because the paired sample is small and skewed. We therefore present the JS divergence results as suggestive rather than confirmatory on **Easy** and **Astro**, and as confirmatory on **Hard**.

E.5 Alternative Evaluation Metrics

E.5.1 Beyond ELPD for Time-Series

Estimating the Expected Log Pointwise Predictive Density (ELPD) for Gaussian Process based Time Series models in `PyMC` is computationally demanding because it necessitates full MCMC sampling. Unlike simpler parametric models, GPs require the inversion of an $n \times n$ covariance matrix - a process with $O(n^3)$ complexity - at every gradient evaluation within the NUTS sampler. To mitigate this cubic scaling bottleneck, we employ fixed-space sub-sampling. By selecting a representative subset

Table 7: Distribution fitting: posterior quality metrics by LLM, method, and dataset. ELPD LOO (↑): posterior predictive log-density on held-out data. Bhatt (↓): Bhattacharyya distance between predicted and true distributions. p_{loo} (↓): PSIS-LOO effective number of parameters (lower indicates cleaner posterior geometry). Bold marks the best value within each (LLM, dataset, metric) column.

		Easy			Astro			Hard		
LLM	**Configuration**	ELPD	Bhatt	p_{loo}	ELPD	Bhatt	p_{loo}	ELPD	Bhatt	p_{loo}
GPT 5.4-mini	PyVision	−1981.48	0.68	4.41	181.31	0.54	6.58	−2426.22	0.40	17.11
	Box-LM	**−1788.34**	0.51	4.68	**281.73**	0.41	6.23	−2381.70	0.30	12.23
	VESTA-None	−1964.96	0.44	5.52	267.33	0.33	5.54	−2359.47	0.21	**6.94**
	VESTA-Dynamic	−1860.53	0.43	3.17	279.21	0.36	5.77	−2365.76	0.18	7.19
	VESTA-Expert	−1853.66	**0.41**	**3.08**	281.12	**0.28**	**5.33**	**−2344.45**	**0.17**	7.17
Claude Sonnet 4.6	PyVision	−1997.60	0.49	2.32	249.36	0.57	2.61	−2661.05	0.44	23.36
	Box-LM	**−1707.01**	0.54	3.35	199.42	0.44	6.67	**−2288.47**	0.27	11.79
	VESTA-None	−1856.27	0.51	**1.92**	**318.42**	0.57	**2.59**	−2330.62	**0.17**	**7.75**
	VESTA-Dynamic	−1848.46	0.97	4.22	291.38	0.32	3.67	−2348.25	0.18	7.82
	VESTA-Expert	−1883.78	**0.44**	2.82	286.56	**0.31**	5.05	−2340.52	0.18	13.85
Kimi K2.5	PyVision	−2133.54	0.54	16.10	180.66	0.54	8.59	−2458.90	0.34	20.45
	Box-LM	**−1788.34**	0.51	4.68	281.73	0.41	6.23	−2381.70	0.30	12.23
	VESTA-None	−1840.80	1.13	5.01	**417.58**	0.97	**1.32**	**−2335.40**	0.24	10.42
	VESTA-Dynamic	−1891.31	**0.50**	5.19	299.67	0.34	6.91	−2351.49	0.20	**8.88**
	VESTA-Expert	−1839.91	0.70	**4.14**	291.38	**0.23**	6.57	−2349.39	**0.17**	13.76

Table 8: Mean performance comparison across all toolkit configurations and datasets for time series. Metrics: ELPD LOO (higher ↑), R^2 (higher ↑), CRPS (lower ↓); We use PYVISION as our baseline.

		Easy			Astro (Chirp)			Hard		
LLM	**Configuration**	ELPD	R^2	CRPS	ELPD	R^2	CRPS	ELPD	R^2	CRPS
Claude Sonnet 4.6	VESTA No-Toolkit	30.2	0.870	0.1127	10.1	0.675	0.1949	-210.7	0.819	0.6891
	VESTA Dynamic-Toolkit	55.9	0.965	0.0749	23.3	0.842	0.1085	-85.1	0.920	0.3537
	VESTA Expert-Toolkit	82.7	0.981	0.0660	42.3	0.909	0.0798	-71.8	0.938	0.3689
	PYVISION	-62.5	≈ 0	0.490	-153.1	≈ 0	0.5052	-335.9	≈ 0	4.833
	BOXLM	**-561.0**	0.639	0.235	-87.6	0.342	0.3318	**-608.6**	0.573	1.1383
GPT 5.4 mini	VESTA No-Toolkit	33.7	0.769	0.1864	26.2	0.791	0.1238	-233.4	0.830	0.6218
	VESTA Dynamic-Toolkit	34.4	0.812	0.1244	44.5	0.882	0.0875	-149.4	0.844	0.4746
	VESTA Expert-Toolkit	62.3	0.962	0.0702	47.5	0.891	0.0811	-117.7	0.948	0.4445
	PYVISION	-103.4	0.085	0.512	-116.5	0.031	0.448	-263.4	≈ 0	4.4379
	BOXLM	-30.6	0.434	0.3438	-88.6	0.024	0.4523	-194.2	0.412	1.5097
Kimi K2.5	VESTA No-Toolkit	80.7	0.830	0.1481	36.6	0.857	0.1061	-222.3	0.741	0.8551
	VESTA Dynamic-Toolkit	98.7	0.909	0.0936	34.6	0.884	0.1001	-114.1	0.907	0.3651
	VESTA Expert-Toolkit	108.1	0.962	0.0720	55.5	0.965	0.0559	-90.1	0.969	0.2776
	PYVISION	-101.1	≈ 0	0.512	-136.7	≈ 0	0.4899	-309.5	≈ 0	4.0653
	BOXLM	-61.6	0.462	0.3249	-91.1	0.004	0.4575	-237.1	0.401	1.7423

of the time series data at regular intervals, we reduce the effective n and this significantly speeds up the evaluation. For consistency, we measure the R^2, CRPS are calculated in the same manner.

For the baseline evaluations, BoxLM generally performs better than PyVision, with the exception of a small number of anomalous cases in Claude-generated PyMC code. Specifically, a few extremely large ELPD values substantially inflate the mean performance metrics. Excluding these 7 outliers (out of 630 total samples) reduces the mean ELPD for Claude Sonnet 4.6 + BoxLM to **40.4** on the Easy dataset and **-150.6** on the Hard dataset, bringing performance closer for Easy, and ahead for Hard dataset to the corresponding No-Toolkit configuration.

In contrast, PyVision exhibits substantially higher failure rates during model fitting. Across datasets, approximately 40–60% of generated samples fail either due to code execution issues or unsuccessful MCMC fitting, representing the highest failure rate among all evaluated approaches. In the remaining successful runs, the fitted models frequently collapse to near mean-value predictions, resulting in poor overall ELPD and near-zero R^2 values. These failures are largely attributable to PyVision's implementation strategy, which is primarily optimized around MAP-based fitting and does not consistently translate to stable MCMC-based inference.

E.5.2 Beyond JS-Divergence for IMFs

For IMF fitting, astronomers often care more whether the recovered power-law slopes and break masses are correct as opposed to the exact distribution function. To stay consistent with the other two distribution fitting tasks, we evaluated the Astro split with Jensen-Shannon divergence in the main results. Here we add an evaluation metric for the Astro split by inferring slope parameters from the predicted PyMC models. We report each VESTA configuration with the Claude Sonnet 4.6 backbone.

The central parameter we study is the logarithmic slope

$$\Gamma(m) = -\frac{d \log f(m)}{d \log m}, \tag{4}$$

which equals α exactly on a power-law segment, but remains well-defined for any fitted family (including fits with no power-law component). From Γ we derive an effective slope parameter α_{eff} using weighted least squares of $\log f$ on $\log m$ over the high-mass window ($m > 1\,M_\odot$), computed identically on the fitted and ground-truth IMFs. We define the α_{eff} **Median Absolute Error** metric on the Astro Distribution Fitting task as the "Median absolute error in the effective high-mass slope against the ground-truth IMF".

We find that results are consistent with Jensen-Shannon divergence, but this metric tends to be more favorable for VESTA-Dynamic. VESTA-Dynamic recovers the Salpeter slope over high masses in nearly all cases ($|\Delta\alpha_{\text{eff}}| = 0.030$ on Salpeter instances). VESTA with no toolkit fails badly on the same instances ($|\Delta\alpha_{\text{eff}}| = 6.21$), with only 14% of fits containing a power-law tail at all. Results for all IMF families are shown in Table 9.

Table 9: Median absolute error in α_{eff} over the high-mass window ($m > 1\,M_\odot$). Lower is better; best per row in bold.

IMF Family	VESTA-None	VESTA-Expert	VESTA-Dynamic
Salpeter	6.213	0.899	**0.030**
Kroupa	**1.171**	1.345	1.343
Chabrier	6.491	**1.367**	4.262
Freeform (tight)	1.472	**1.248**	1.288
Freeform (wide)	1.298	0.859	**0.805**

E.6 Impact of the Iteration Budget in VESTA

All VESTA variants are run with a fixed iteration budget of $N = 5$. In this section, we examine where the best performing models occur in this fixed budget. We study this to answer whether VESTA-Expert outperforming VESTA-Dynamic could be an artifact of expert tools being simpler to interpret in fewer steps.

Step 0 is the zero-shot proposal with no refinement, and Step i refers to the i^{th} step in the VESTA iterative procedure. For each instance, we identify the step with the lowest AIC among VESTA's 5 candidates and the zero-shot baseline, and report the percentage of instances for which the best PyMC model occurred at each step. Results for all three toolkit modes and backbone models are shown in Table 10.

Zero-shot proposals rarely produce the best model. A zero-shot baseline provides the best model in only 3-10% of cases, even with the VESTA-None variant where no tools are utilized. With VESTA-Dynamic, the number of cases where the zero-shot proposal improves over VESTA is reduced further to 2-5%.

If VESTA-Dynamic had a higher concentration of best-performing models at Step 5, that would indicate that the complex generated tools need more iterations to process successfully. We find the opposite. Across all three backbone models, VESTA-Dynamic has the lowest share of best models at the final index. The starkest case is Kimi-2.5, where 82/100 of the best PyMC models occur at Step 5 with VESTA-Expert, but only 48/100 with VESTA-Dynamic. This indicates that VESTA-Expert is potentially more constrained by the $N = 5$ budget than VESTA-Dynamic. Further, VESTA-Dynamic tends to converge earlier, with a spike at Step 1 that is absent for VESTA-Expert.

Table 10: Percentage of instances for which the best PyMC model (lowest AIC) occurred at each step. Step 0 is the zero-shot proposal; Steps 1-5 are VESTA refinement iterations.

	VESTA-None			VESTA-Expert			VESTA-Dynamic		
Step	Claude	GPT	Kimi	Claude	GPT	Kimi	Claude	GPT	Kimi
0 (0-Shot)	10	3	8	8	2	10	2	4	5
1	3	9	1	1	5	1	14	16	17
2	1	7	0	2	5	3	7	3	10
3	2	4	2	4	2	3	6	3	9
4	1	1	2	6	3	1	2	1	11
5	83	76	87	79	83	82	69	73	48

We argue that these observations are sensible given the structure of the toolkits. The generated tools in VESTA-Dynamic are predominantly multi-panel and deliver a large amount of information in a single call. In contrast, Expert tools are designed to perform a single function and must be composed across several iterations. Compared to using no tools, VESTA-Dynamic also converges to the best solution faster. For Kimi-2.5, the number of best models at Step 5 is reduced from 87/100 to 48/100 when the agent is allowed to dynamically generate tools.

This analysis shows that (1) a zero-shot setting rarely produces a better model than VESTA, and (2) equipping VESTA with dynamic tool generation helps agents find optimal solutions at earlier iterations. Consequently, the finding that VESTA-Expert outperforms VESTA-Dynamic in most cases is unlikely to be an artifact of the iteration budget. The gap is also not explained by dynamic tools failing to improve PyMC model predictions, since VESTA-Dynamic rarely has its best model at Step 0. We note that all claims in this paper hold under a fixed and equal iteration budget of $N = 5$.

F Expert Tools in Detail

F.1 Distribution Fitting

1. **CalculateMoments**: Computes the mean, variance, skewness, and excess kurtosis of the input data. Returns a JSON artifact with a plain-language interpretation to guide distribution selection, including symmetry hints (e.g., right-skewed data suggests Gamma, Lognormal, or Weibull families) and tail-weight hints (e.g., leptokurtic data suggests Student-t, Cauchy, or Laplace families).
2. **Histogram**: Plots a histogram of the empirical data with the fitted distribution's probability density function (PDF) overlaid. Handles both single distributions and mixtures by summing component PDFs weighted by their mixture weights. Provides an immediate visual check of whether the model captures the overall shape, modality, and spread of the data. When no model has been fitted yet, produces a raw histogram without an overlay.
3. **SegmentDistributionsAndCalculateMoments**: Segments the data into a specified number of mixture components using a Gaussian Mixture Model (GMM), then computes per-component moments (mean, variance, skewness, kurtosis). Produces both a segmentation image with a total mixture overlay and a JSON summary of per-component statistics with distribution family hints. Most useful when the histogram appears multimodal.
4. **QQPlot**: Generates a Quantile-Quantile (Q-Q) plot comparing empirical data quantiles to theoretical quantiles from the currently fitted distribution. Linearity indicates a good fit; S-shaped curvature signals tail mismatch; one-sided curvature suggests skew; and sharp tail departures may indicate outliers or heavier tails than the model captures.
5. **PlotTailsTransform**: Produces log-log and semi-log complementary CDF (CCDF) plots to diagnose tail behavior. A straight line on the log-log plot indicates power-law or Pareto-type heavy tails, while a straight line on the semi-log plot indicates exponential decay. Useful for distinguishing heavy-tailed from light-tailed distributions when the histogram alone is ambiguous.
6. **ProbabilityPlot**: Generates a probability plot comparing the empirical CDF to the fitted distribution's theoretical CDF. A consistent horizontal shift indicates a mis-specified location

Table 11: Per-sample tool coverage: percentage of samples in which each Expert tool is invoked at least once during the iterative refinement loop.

Tool	Easy	Astro (Chirp)	Hard	Overall
`get_dominant_period`	100.0%	100.0%	99.1%	99.5%
`fit_vs_actuals_w/_resid.`	100.0%	96.0%	48.2%	71.9%
`residuals_auto_corr._plot`	2.0%	10.0%	54.5%	31.4%
`residuals_auto_corr._score`	0.0%	0.0%	6.4%	3.3%
`fit_vs_actuals (default at step-0)`	100.0%	100.0%	100%	100%

parameter; a slope mismatch indicates a scale misfit; and systematic tail deviations suggest distributional misfit. Also reports a Kolmogorov-Smirnov (KS) statistic for quantitative goodness-of-fit assessment.

F.2 Time Series

1. **GetDominantPeriod**: Extracts the dominant period from the time series using Fast Fourier Transform (FFT) analysis. Returns a plain-text summary of the detected period. Most useful when Periodic or PeriodicComplex kernels are under consideration and the period has not yet been numerically determined. The result is available in the subsequent feedback iteration.

2. **FitVsActuals**: Produces a visual overlay of the Gaussian Process (GP) fit on the raw time series data. Essential for visually assessing whether the model adequately captures the underlying trend and seasonality while appropriately discounting noise. Falls back to a raw series plot if no model has been fitted yet.

3. **FitVsActualsWithResidualsDistribution**: Generates a combined plot showing the GP fit overlaid on the observed time series alongside the distribution of residuals. Used to assess whether residuals resemble white noise; a broadly normal residual distribution is indicative of a well-specified model. Falls back to a raw series plot if no model has been fitted yet.

4. **ResidualsAutoCorrelationPlot**: Produces an Autocorrelation Function (ACF) plot of the model residuals to check for temporal independence. Significant spikes above the confidence band indicate that the model is failing to capture some latent structure in the data. Falls back to a raw series plot if no model has been fitted yet.

5. **ResidualsAutoCorrelationScore**: Applies the Ljung-Box statistical test to assess residual independence numerically. A p-value greater than 0.05 indicates that the residuals are consistent with white noise, suggesting an adequate model fit. Returns a plain-text summary and requires a fitted model to produce meaningful results.

F.2.1 Feedback Modality and Adaptive Selection

The static and dynamic toolkits differ fundamentally in their feedback modality. The expert toolkit provides narrow, targeted signals—such as returning a single scalar for dominant frequency—which allows the VLM to rapidly integrate findings and converge quickly on structurally simple datasets (Easy, Chirp).

In contrast, the dynamic toolkit generates comprehensive, multi-panel visualizations that synthesize frequency estimation, residual analysis, and harmonic structure into a single artifact. While this rich feedback demands more visual interpretation from the VLM, it is critical for diagnosing complex signals where isolated scalars are insufficient.

As shown in Table 11, the VLM dynamically adapts its tool selection to the underlying problem complexity. Simple goodness-of-fit overlays dominate the early steps for clean periodic datasets. However, when applied to the Hard dataset, the agent heavily prioritizes autocorrelation and residual diagnostics to detect more subtle, persistent temporal dependencies.

F.2.2 Comparison with PyVision Baseline

Apart of choice of modalities, and complexity of serving the tool (via panels), both dynamic-toolkit and expert-toolkit lean into visual feedback. On the other hand, PyVision also generates dynamic tools on the fly, but they diverge significantly in diagnostic focus, output density, and tool lifecycle.

Diagnostic Category Coverage. As detailed in Table 12, both systems heavily utilize Lag/Autocorrelation and Frequency Estimation (∼85–94%). However, VESTA's dynamic toolkit exhibits a massive preference for Residual Analysis (91.9% vs. PyVision's 11.9%) and exclusively employs Statistical Tests (28.6% vs. 0%). Because residual structure is the primary indicator of GP kernel misspecification, this gap directly impacts model selection quality.

Table 12: Diagnostic category coverage (% of samples with ≥1 tool in category). PV = PyVision.

Category	Easy		Astro (Chirp)		Hard		Overall	
	VESTA	PV	VESTA	PV	VESTA	PV	VESTA	PV
Residual Analysis	92.0	22.0	80.0	0.0	97.3	12.7	**91.9**	**11.9**
Lag / Autocorrelation	98.0	98.0	92.0	94.0	84.5	84.5	89.5	90.0
Frequency Estimation	100.0	100.0	100.0	100.0	71.8	89.1	85.2	94.3
Statistical Tests	34.0	0.0	32.0	0.0	24.5	0.0	**28.6**	**0.0**

G Analysis of Generated Tools Not in the Expert Toolkit

G.1 Tool Persistence and Reuse.

A key architectural advantage of VESTA's dynamic-toolkit is the persistence of its generated functions. Within a single problem instance, 79.5% of VESTA's dynamically generated tools are re-invoked across subsequent analysis steps (averaging 1.87 calls per tool; Table 13). This capability is most useful for tools that evaluate model fit, allowing the agent to iteratively reassess model fit using a consistent baseline. Consider the case study shown in Figure 5. As VESTA iteratively refines its distributional hypothesis (from exponential to log-normal), the tool can be re-invoked with an updated PyMC model to regenerate Panel 1's Q-Q plot against the new candidate distribution without incurring additional tool-writing cost. Further, the remaining diagnostic panels act as a consistent reference frame, meaning VESTA does not have to regenerate tools to re-examine the underlying data when evaluating a new distributional hypothesis. PyVision treats generated code as single-use execution cells, entirely lacking a mechanism for revisitation.

Table 13: VESTA tool reuse statistics within single problem instances.

Dataset	Tools	Reused ≥1×	Avg Invocations
Easy	82	64 (78.0%)	1.93
Hard	148	126 (85.1%)	1.75
Astro	73	51 (69.9%)	2.07
Overall	**303**	**241 (79.5%)**	**1.87**

G.2 Distribution Fitting

1. **diagnostic_fit_checks:** Naming a concrete model family (gaussian, gamma, lognormal, Pareto, Weibull, etc.) and trying it on the data. Each family encodes different assumptions. These tools allow for typically allow for a visual comparison of multiple model families at once. Occasionally, we observe some single use model fitting.
2. **information_criteria:** Numerical scores that rank competing fits while penalizing model complexity. Beyond simply fitting and visualizing models, AIC and BIC provide quantitative fit metrics.

Table 14: Analysis of functions in VESTA-generated tools that are not contained in the expert toolkit for Distribution Fitting.

Function	Easy	Hard	Astro	All
Diagnostic Fit Checks	**100.0**	96.9	**100.0**	98.4
Information Criteria	**82.1**	73.8	65.6	73.7
MLE Fitting	69.6	36.2	**67.2**	51.4
Mean Excess Plot	5.4	7.7	**36.1**	14.2
Hill Estimator	0.0	1.5	**6.6**	2.4
Box–Cox	**1.8**	0.0	0.0	0.4
D'Agostino Normal Test	0.0	**0.8**	0.0	0.4
Shapiro–Wilk	0.0	**0.8**	0.0	0.4

Table 15: Analysis of functions in VESTA-generated tools that are not contained in the expert toolkit for Distribution Fitting.

Function	Easy	Hard	Astro
Density Visualization	51.9	**67.6**	33.8
Distribution Shape Summary	38.3	**50.3**	30.9
Moving Average	**30.9**	25.5	27.9
Regime Switching Analysis	2.5	**30.3**	0.0
Heavy Tail Diagnostics	9.9	**20.7**	5.9
QQ Plot	**13.6**	9.7	10.3
Shapiro–Wilk	**17.3**	6.2	11.8
Stationarity Tests	1.2	**11.7**	4.4
Differencing	4.9	**10.3**	2.9
ARIMA	0.0	**5.5**	0.0
Change Point Detection	0.0	**4.1**	1.5
Information Criteria	0.0	**4.8**	0.0
MLE Fitting	0.0	**1.4**	0.0
Kolmogorov–Smirnov	0.0	0.7	**1.5**
Log Transform	0.0	0.7	**1.5**
Jarque–Bera	**1.2**	0.0	0.0
ECDF	0.0	**0.7**	0.0

3. **mle_fitting:** Maximum likelihood estimation: choosing the parameter values (μ, σ, shape, scale, ...) that maximize the probability of observing the data under the chosen family. This is the how you actually of fit models, distinct from what models we want to test in `diagnostic_fit_checks`. MLE gives you the canonical "best" parameters under a given family, plus a likelihood value that feeds directly into AIC/BIC.
4. **mean_excess_plot** Plots the conditional expectation $\mathbb{E}[X - u \mid X > u]$ against threshold u. For the Generalized Pareto distribution this function is linear in u, so a straight line in the upper tail signals a GPD-like tail and tells you where the "extreme regime" begins. This is a tail-diagnostic that complements **diagnostic_fit_checks**. These test help it VESTA decide if the tail behaves as that family predicts and at what threshold the heavy-tail regime kicks in. The 36% rate on Astro vs. $\sim$5% elsewhere is a sign the model recognizes astronomical data as tail-dominated.
5. **hill_estimator** Estimates the tail index α of a heavy-tailed distribution from the largest k order statistics, giving a concrete number for "how heavy" the tail is. A Hill plot ($\hat{\alpha}$ vs. k) lets you check stability and pick a sensible threshold. This refines a Pareto/power-law fit by pinning down its single most important parameter, and serves as a sanity check on MLE-fit tail parameters — if MLE on a full-data Pareto fit and a Hill estimate on the upper tail disagree, the body of the distribution is contaminating your tail estimate.
6. **shapiro_wilk** A formal hypothesis test for whether data come from a normal distribution. If Shapiro-Wilk rejects normality strongly, that rules out the normal family in `diagnostic_fit_checks`.
7. **box_cox** A parametric family of power transforms $y = (x^\lambda - 1)/\lambda$ that searches for the λ making the transformed data closest to normal and can be useful when working with exotic, heavy-tailed distributions.

G.3 Time Series

1. **density_visualization:** Overlays a histogram with a kernel density estimate (KDE) to give a non-parametric picture of the marginal distribution of a time series. This is typically the first diagnostic VESTA reaches for: before committing to any parametric family, the histogram/KDE pair reveals modality, skew, and gross tail behavior at a glance. The KDE bandwidth controls the bias–variance tradeoff in smoothing; narrower bandwidths reveal fine structure while wider ones expose the global shape.
2. **distribution_shape_summary:** Computes the first four standardized moments (mean, variance, skewness, and excess kurtosis) along with the coefficient of variation $\text{CV} = \sigma/\mu$. These scalars compress the shape of an entire distribution into a small, comparable summary. Skewness distinguishes asymmetric processes (e.g. log-normal dwell times) from symmetric ones. Kurtosis flags heavy tails before any formal test is run. This is the numerical complement to **density_visualization**.
3. **moving_average:** Computes a rolling mean $\bar{x}_t = \frac{1}{w}\sum_{i=0}^{w-1} x_{t-i}$ over a sliding window of width w. This is low-pass filter suppresses high-frequency noise and reveals the underlying trend or long-timescale signal. VESTA uses it both as a standalone smoother and as a preprocessing step before stationarity tests or ARIMA fitting, since many parametric models assume a trend-free series.
4. **regime_switching_analysis:** A grouped capability covering Gaussian Mixture Models (GMMs), bimodality and multimodality detection, k-means clustering on temporal states, Hidden Markov Models (HMMs), and log-normal fits to dwell times. Together these tools test the amount of time a signal spends in qualitatively distinct states?, GMMs and k-means partition the amplitude space into clusters; HMMs additionally impose a Markov transition structure on state sequences; dwell-time fits characterize how long the process stays in each regime. This matters for telegraph-noise or switching processes where a single stationary distribution is a category error. The high rate on **Hard** reflects that harder series are more likely to be genuine multi-state processes.
5. **heavy_tail_diagnostics:** A grouped capability covering Pareto/power-law identification, Generalized Extreme Value (GEV) fits, Hill estimator calls, and mean-excess plots; see the analogous entries in the distribution-fitting appendix for mathematical detail on the latter two. In the time series context, VESTA applies these when the marginal distribution of the series itself (rather than residuals) appears heavy-tailed, e.g. burst-like signals or processes with rare but extreme excursions.
6. **qq_plot:** Plots empirical quantiles $F_n^{-1}(p)$ against theoretical quantiles $F_\theta^{-1}(p)$ for a candidate family F_θ. These are typically used to measure the distribution of residuals.
7. **shapiro_wilk:** A formal hypothesis test of the null that the data are drawn from a normal distribution, with test statistic $W = (\sum a_i x_{(i)})^2 / \sum (x_i - \bar{x})^2$ where $x_{(i)}$ are order statistics and a_i are weights derived from the expected normal order statistics. In the time series context, VESTA typically applies this to residuals after trend/seasonality removal to verify that the generating process is Gaussian, which is a prerequisite for classical ARIMA inference.
8. **stationarity_tests:** A grouped capability covering the Augmented Dickey–Fuller (ADF) test for a unit root and the KPSS test for level/trend stationarity. These are complementary. ADF tests the null of a unit root while KPSS tests the null of stationarity, so using both guards against the failure mode of each test individually. Stationarity is a prerequisite for most parametric time series models; this capability gates whether VESTA proceeds to ARIMA fitting or first applies differencing or detrending.
9. **differencing:** Computes the lag-d difference $\nabla^d x_t = x_t - x_{t-d}$, most commonly $d = 1$, the first difference. This is the standard cure for a unit root: if the ADF test fails to reject non-stationarity, first-differencing typically removes a stochastic trend and produces a stationary series suitable for ARIMA fitting. Seasonal differencing ($d = s$, the period) removes periodic non-stationarity. VESTA uses this as a preprocessing step that is directly triggered by stationarity test results.
10. **arima:** Fits an Autoregressive Integrated Moving Average model $\phi(B)(1-B)^d x_t = \theta(B)\epsilon_t$, where B is the backshift operator, p autoregressive terms capture dependence on recent values, d differences remove trends, and q moving-average terms model the

residual autocorrelation structure. ARIMA is the workhorse parametric model for univariate stationary time series; it is typically applied after stationarity tests confirm (or differencing achieves) stationarity, and model order (p, d, q) is selected via AIC/BIC.

11. **change_point_detection:** Identifies times τ at which the statistical properties of the series, mean, variance, or autocorrelation structure, shift abruptly. Unlike **regime_switching_analysis**, which models soft probabilistic membership in states, change-point methods partition the timeline into contiguous segments and locate the breakpoints. This is appropriate when the series undergoes irreversible structural shifts (e.g. a one-time intervention) rather than recurrent toggling between states.
12. **information_criteria:** Computes AIC $= 2k - 2\hat{\ell}$ and BIC $= k \ln n - 2\hat{\ell}$, where k is the number of free parameters, n the sample size, and $\hat{\ell}$ the maximized log-likelihood. These penalize model complexity to prevent overfitting and give you a principled basis for comparing models of different orders (e.g. ARIMA$(1, 1, 1)$ vs. ARIMA$(2, 1, 2)$) or different parametric families on the same data. AIC favors predictive accuracy while BIC applies a stronger complexity penalty and is consistent for model selection.
13. **mle_fitting:** See the analogous entry in the distribution-fitting appendix. In the time series context, MLE is used both for fitting marginal distributions (e.g. the amplitude distribution of a stationary series) and for estimating ARIMA parameters via conditional or exact likelihood. The resulting likelihood value feeds directly into AIC/BIC for model comparison.
14. **kolmogorov_smirnov:** A nonparametric goodness-of-fit test based on the supremum distance $D_n = \sup_x |F_n(x) - F_0(x)|$ between the empirical CDF and a hypothesized distribution F_0. Unlike Shapiro–Wilk, the KS test is distribution-free and applies to any fully specified F_0, making it useful when VESTA wants to test an arbitrary fitted family rather than only normality. It is less powerful than Anderson–Darling in the tails but gives a straightforward visual diagnostic via the CDF overlay.
15. **log_transform:** Applies $y_t = \log(x_t)$ to stabilize variance in series whose standard deviation grows with the mean, a signature of log-normal or multiplicative processes. Log-transforming a series with exponential growth or multiplicative noise often produces an approximately Gaussian, stationary series amenable to ARIMA. It is also a prerequisite for power-law tail analysis, where log–log linearity of the CCDF is the diagnostic signature.
16. **jarque_bera:** A moment-based normality test with statistic $JB = \frac{n}{6}\left(\gamma_1^2 + \frac{\gamma_2^2}{4}\right)$, which is asymptotically $\chi^2(2)$ under normality. Because it is constructed directly from skewness and kurtosis, it is more powerful than the KS test against symmetric heavy-tailed alternatives (large γ_2) and skewed alternatives (large $|\gamma_1|$). In practice VESTA uses it as a fast scalar complement to the full **distribution_shape_summary**.
17. **ecdf:** Plots the empirical cumulative distribution function $F_n(x) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}[x_i \leq x]$. The ECDF is a non-parametric, unsmoothed estimate of the true CDF and serves as the ground truth against which parametric fits are visually compared. It is the natural complement to the KS test: the test statistic is the maximum vertical gap between the ECDF and the hypothesized CDF curve.

H Runtime of VESTA

I Prompts

This section contains all the prompts used by our method and baselines.

J Limitations

While VESTA demonstrates strong performance across DAWN, several limitations remain. First, VESTA relies on VLMs to interpret multi-panel diagnostic visualizations, and current VLMs struggle to fully unpack and reason over complex visual outputs. This bottleneck directly limits the gains from dynamic tool creation, as the most sophisticated generated tools contain multi-panel images

Table 16: Runtimes by Domain, Dataset, Mode, and Model

Domain	Dataset	Mode	Model	n	Avg Time (s)	Avg Steps	Total Time (s)
df	imf	genonly	claude_sonnet46	50	84.8	4.1	4238.9
			gpt54_mini	50	163.5	5.1	8173.1
			kimi25	50	1122.6	5.0	56131.8
		none	claude_sonnet46	50	64.9	4.4	3242.6
			gpt54_mini	50	31.6	5.4	1577.9
			kimi25	50	1073.9	5.3	53694.8
		static	claude_sonnet46	50	135.2	5.8	6762.3
			gpt54_mini	50	40.1	5.8	2004.7
			kimi25	50	1172.4	5.5	58622.3
	mixed	genonly	claude_sonnet46	100	125.0	5.0	12503.9
			gpt54_mini	100	91.6	5.0	9163.7
			kimi25	100	1272.7	4.5	127267.2
		none	claude_sonnet46	100	103.3	5.3	10333.3
			gpt54_mini	100	21.4	5.2	2142.7
			kimi25	100	29.0	5.5	2905.0
		static	claude_sonnet46	100	152.5	5.4	15245.5
			gpt54_mini	100	24.4	5.5	2443.5
			kimi25	100	31.3	5.3	3127.5
	single	genonly	claude_sonnet46	50	78.3	3.7	3914.8
			gpt54_mini	50	19.4	4.6	967.9
			kimi25	50	1166.1	3.9	58306.0
		none	claude_sonnet46	50	88.6	5.2	4427.7
			gpt54_mini	50	15.6	4.2	782.2
			kimi25	50	24.6	5.5	1232.1
		static	claude_sonnet46	50	101.8	4.8	5088.8
			gpt54_mini	50	26.7	4.7	1335.1
			kimi25	50	27.5	5.5	1375.1
ts	easy	genonly	claude_sonnet46	50	132.3	4.8	6614.0
			gpt54_mini	50	14.8	3.3	740.8
			kimi25	50	883.6	3.8	44181.9
		none	claude_sonnet46	50	51.5	3.2	2576.6
			gpt54_mini	50	11.5	3.1	576.8
			kimi25	50	40.1	5.9	2006.1
		static	claude_sonnet46	50	38.6	2.4	1927.9
			gpt54_mini	50	17.2	3.3	861.9
			kimi25	50	50.5	5.8	2527.5
	gravitational_chirp	genonly	claude_sonnet46	50	153.0	4.5	7648.1
			gpt54_mini	50	96.7	3.3	4834.6
			kimi25	50	986.0	4.1	49298.0
		none	claude_sonnet46	50	62.6	3.5	3132.0
			gpt54_mini	50	17.0	3.4	852.0
			kimi25	50	39.8	5.7	1989.7
		static	claude_sonnet46	50	87.2	3.4	4358.0
			gpt54_mini	50	22.0	3.4	1099.4
			kimi25	50	43.7	5.9	2184.9
	medium	genonly	claude_sonnet46	110	99.1	3.6	10902.3
			gpt54_mini	110	136.1	4.2	14968.5
			kimi25	110	1166.3	4.5	128297.0
		none	claude_sonnet46	110	53.1	3.2	5842.3
			gpt54_mini	110	12.8	2.8	1409.4
			kimi25	110	38.3	5.7	4208.5
		static	claude_sonnet46	110	49.5	2.6	5443.2
			gpt54_mini	110	15.9	2.7	1750.4
			kimi25	110	43.6	5.8	4791.9

that compose multiple functions. This gap likely accounts for the remaining performance differential between VESTA-Dynamic and VESTA-Expert.

Second, the iterative nature of VESTA incurs non-trivial computational cost, as each iteration requires fitting up to p PyMC models via MAP estimation. This limits scalability to very large datasets.

Finally, while DAWN covers two domains central to scientific modeling and includes real-world **Astro** tasks, the benchmark is synthetically generated. Performance on fully observational scientific

VESTA **Time Series Critique Prompt**

You are the VLM critic of VESTA. Inspect the latest fit of the current Gaussian-process PyMC model and decide whether to refine the kernel composition further or to terminate the iterative refinement loop. The prompt covers stationary periodic structure as well as non-stationary (warped periodic) wave signals.

Goal: Determine whether the current GP kernel composition correctly captures the trend, seasonality, and non-stationarity of the series, and whether the priors are well-scaled for the observed amplitude and period.

Current Model Information
- Current PyMC code: `{current_pymc_code}`
- Current kernels: `{current_kernels}`
- Previously tested kernel combinations: `{tested_kernels}`
- History of previously executed tools: `{tool_history}`

Iteration Summaries: `{summaries}`

Important Constraints
- Evaluate fit quality based on the most recent iteration's tool output and visual feedback.
- Take the trajectory of past iterations and their metric values into account.
- You may revisit a previously tested kernel combination, but you must use sufficiently different priors.
- Do not invoke the same tool more than twice across the run.
- Combine kernels by addition only (`cov = cov1 + cov2`).
- `warped_periodic` subsumes `periodic`; do not combine them.

Stationary vs. Non-Stationary Wave Check: [visual rule for deciding whether cycle spacing is constant (stationary) or sweeping (non-stationary), and what kernel that implies]

Kernel Family Guide: [valid kernel names, when each kernel applies, and how kernels may be combined are listed here]

Prior Specification Rules: [recommended prior families and scales for length-scale, amplitude, period, warp coefficients, and observation noise are listed here, with anti-patterns for misuse of `pm.Deterministic` and undefined initializer variables]

Proposal Diversity Rules: [the primary proposal must be the best guess; the remaining proposals must explore structurally different kernel combinations, including at least one with and one without `warped_periodic` when non-stationary periodicity is suspected]

Diagnostic Results: `{tool_outputs}`

Task: If the current fit is satisfactory, terminate the loop early by including the literal token `COMPLETE` in your description. Otherwise, propose exactly `{num_proposals}` diverse revised models following the rules above. State explicitly whether the stationary-vs-non-stationary wave check passes and, if non-stationary, which direction the frequency sweeps.

Output Format: Return **only** a valid JSON object with `description` and `proposals`. Each proposal must include `kernels` and `priors`. Never output anything outside the JSON object.

Figure 7: Critique-stage prompt used by VESTA for time series modeling.

datasets, where ground-truth parameters are unknown, remains an important avenue for future evaluation.

VESTA **Distribution Fitting Critique Prompt**

You are the VLM critic of VESTA. Inspect the latest fit of the current PyMC model and decide whether to refine it further or to terminate the iterative refinement loop.

Goal: Determine whether the current distribution model correctly captures the shape, location, scale, and tail behavior of the data.

Current Model Information
- Current PyMC code: `{current_pymc_code}`
- Current distribution family: `{current_family}`
- Previously tested distribution families: `{tested_families}`
- History of previously executed tools: `{tool_history}`

Iteration Summaries: `{summaries}`

Important Constraints
- Evaluate fit quality based on the most recent iteration's tool output and visual feedback.
- Take the trajectory of past iterations and their metric values into account.
- You may revisit a previously tested family, but you must use sufficiently different priors.
- Do not invoke the same tool more than twice across the run.
- Prefer interpretable distributions and avoid unnecessary complexity.

Model Fit Diagnostics: [shape capture, peak alignment, tail behavior, mixture detection, and overfitting checks are listed here]

Distribution Family Guide: [valid single families, mixture syntax, parametrization reference for each family, support-validation rules, and family-specific prior constraints are listed here]

Prior Specification Rules: [instructions for reading the histogram to extract peak/spread/support/tails, translating those observations into hyperparameter values, and the variable naming convention for single and mixture priors are listed here]

Proposal Diversity Rules: [the primary proposal must be the best guess; the remaining proposals must be structurally different alternatives covering both single distributions and mixtures]

Diagnostic Results: `{tool_outputs}`

Task: If the current fit is satisfactory, terminate the loop early by including the literal token `COMPLETE` in your description. Otherwise, propose exactly `{num_proposals}` diverse revised models following the rules above.

Output Format: Return **only** a valid JSON object with `description` and `proposals`. Each proposal must include `distribution_family`, `is_mixture`, and `priors`. Never output anything outside the JSON object.

Figure 8: Critique-stage prompt used by VESTA for distribution fitting.

VESTA **Tool Selection Prompt**

You are evaluating the current PyMC model fit. The latest fit is shown in the attached plot.

Tool Registry: `{tool_descriptions}`

Decision Rules
- Select exactly one tool from the registry that best diagnoses the current fit.
- Prefer tools that probe the suspected weakness of the current model: `{model_weakness_examples}`.
- If no existing tool in the registry is adequate, return the `generate_new_tool` action and describe what the new tool should compute; the tool-creation step will then code it.
- You may not invoke the same tool more than twice across the run.

Output Format: Return **only** a valid JSON object naming the selected tool and any arguments it requires.

Domain-specific values for the placeholders above:
- **Distribution fitting** - `{model_weakness_examples}` = `peak misalignment, tail mismatch, multimodality, support violation`.
- **Time series modeling** - `{model_weakness_examples}` = `missed period, residual structure, amplitude envelope, frequency sweep, anomalies`.

Figure 9: Tool-selection prompt used by the Generate-Tools stage of VESTA.

VESTA **Tool Creation Prompt**

You are creating a new diagnostic tool that will be added to the tool registry of VESTA and may be reused in later iterations of this run. The tool is a Python function executed in a sandbox with access to: `numpy as np`, `pandas as pd`, `scipy.stats`, `matplotlib.pyplot as plt`, `sklearn{extra_imports}`.

Function Signature
- Define a single function: `def tool(data: {data_handle}, map_estimate: dict, family_or_kernel: str) -> str`
- The return value is the path to a saved figure (PNG) or a short textual summary that the VLM critic can read.

Implementation Rules
- Begin directly with the `def tool(...)` block. Do **not** include import statements; required modules are already in scope.
- Use `map_estimate` to construct a comparison between the observed data and the current model.
- Save any figure with `plt.savefig(...)` and return its path; do not call `plt.show()`.
- The tool must run end-to-end without user interaction.
- `{domain_specific_code_rule}`

Output Format: Return **only** a JSON-parseable dict with key `"code"` containing the function as a single-line string with `\n` for newlines.

Domain-specific values for the placeholders above:
- **Distribution fitting** - `{extra_imports}` = (none beyond the default sandbox); `{data_handle}` = `np.ndarray`; `{domain_specific_code_rule}` = "Visualize the empirical and model-implied densities or transformed-data summaries; do **not** fit a new PyMC model inside the tool."
- **Time series modeling** - `{extra_imports}` = `, statsmodels.api as sm, scipy.signal`; `{data_handle}` = `pd.Series with a DatetimeIndex`; `{domain_specific_code_rule}` = "Use the time index for any plot's x-axis and align predictions with `data.index`; do **not** re-fit the GP inside the tool."

Figure 10: Tool-creation prompt used by the Generate-Tools stage of VESTA.

VESTA **Summarization Prompt**

You are summarizing the latest iteration of a {run_kind} run so that the next iteration of VESTA can reason over the full refinement trajectory without consuming the raw intermediate outputs.

Inputs Available
- The visual description produced for the current iteration.
- The best PyMC model code and its fitted parameters (map_estimate).
- The metric value and any auxiliary fit statistics.
- The output (image or text) produced by the selected or newly created tool.

What the Summary Must Contain
- A short textual description of what the data look like and what structure the current best model captures.
- The chosen {model_field_name} and its key fitted parameters as concise key–value pairs.
- The metric value for the current best model.
- One or two sentences describing what the tool output revealed, focusing on remaining mismatch between the model and the data.
- {description_specialization}

Style Rules
- Be concrete: name parameters and numeric values rather than describing them abstractly.
- Do **not** re-derive the model or restate prompt instructions.
- Keep the summary self-contained; it replaces the raw per-iteration outputs in subsequent iterations.

Output Format: Return a JSON object with fields description, {model_field_name}, map_estimate, metric, and tool_output_summary.

Domain-specific values for the placeholders above:
- **Distribution fitting** - {run_kind} = distribution fitting; {model_field_name} = distribution_family; {description_specialization} = "Mention any tail, peak, or modality features the current model fails to reproduce."
- **Time series modeling** - {run_kind} = time series modeling; {model_field_name} = kernels; {description_specialization} = "Mention any trend, seasonality, or non-stationarity features the current kernels fail to reproduce."

Figure 11: Prompt used by the Summarize stage of VESTA.

BoxLM Proposal Prompt for Time Series Modeling

You are a brilliant statistician modeling a time series dataset using Gaussian processes. Your job is to come up with a GP model that explains the time series by writing a PyMC probabilistic program. In the first round you receive only the dataset description; in later rounds you also receive the previous program along with the critic's hypotheses and synthesis.

Inputs
- Dataset description: `{dataset_description}`
- Column description: `{column_description}`
- Previous round's hypotheses: `{prev_hypotheses}`
- Previous round's synthesis: `{prev_synthesis}`

Kernel Family Guide
- `pm.gp.cov.Linear` - persistent upward or downward drift.
- `pm.gp.cov.Periodic` - smooth repeating sinusoidal patterns.
- `pm.gp.cov.ExpQuad` (RBF) - extremely smooth curves.
- `pm.gp.cov.Matern52` - rough or jagged paths.
- Combine kernels by addition for independent structures, by multiplication for modulation. Do not combine RBF and Matern in the same model.

PyMC API Rules
- Import `pymc` (not `pymc3`). Do not use `sd` as an argument; use `sigma` instead.
- Use the variable name `y_obs` for the likelihood.
- Index columns explicitly from `observed_data`; never use `observed_data.index` to grab raw data.
- The model must use Gaussian processes (`pm.gp`); plain regression is not allowed.
- Do not use `pm.Constant` for parameters and do not pass `dims="obs_id"` for the GP marginal likelihood.

Required Function Skeleton
- Define a function `def gen_model(observed_data)` that loads data via `observed_data["column_name"].values` (never a placeholder array literal).
- Before the model block, normalize the time axis to `[0, 1]` and center the target values.
- Inside `with pm.Model() as model:` declare the kernel priors, build the covariance, define `gp = pm.gp.Marginal(cov_func=cov)`, and set the likelihood `y_obs = gp.marginal_likelihood('y_obs', ..., sigma=sigma)`.
- Sample with `trace = pm.sample(1000, tune=500, target_accept=0.90, chains=3, cores=1, random_seed=rng1, idata_kwargs={"log_likelihood": True})`.
- Then run `posterior_predictive = pm.sample_posterior_predictive(trace, random_seed=rng2, return_inferencedata=False)`.
- Return `(model, posterior_predictive, trace)` in that order.

Response Structure
- First, sketch a high-level GP model for the data and explicitly cite which previous-round hypotheses (if any) the sketch addresses.
- Then write the full program in a Python code block, with comments on lines that address specific hypotheses.

Figure 12: Proposal prompt used by the BoxLM baseline for time series modeling.

BoxLM Proposal Prompt for Distribution Fitting

You are a brilliant statistician modeling a dataset. Your job is to come up with a generative model that explains the true data by writing a PyMC probabilistic program. In the first round you receive only the dataset description; in later rounds you also receive the previous program along with the critic's hypotheses and synthesis.

Inputs
- Dataset description: `{dataset_description}`
- Column description: `{column_description}`
- Previous round's hypotheses: `{prev_hypotheses}`
- Previous round's synthesis: `{prev_synthesis}`

PyMC API Rules
- Import `pymc` (not `pymc3`). Do not use `sd` as an argument; use `sigma` instead.
- Use the variable name `y_obs` for the likelihood.
- Index columns explicitly from `observed_data`; never use `observed_data.index` to grab raw data.
- Wrap every input feature in a `pm.MutableData` container with `dims="obs_id"`, and pass `dims="obs_id"` to the likelihood as well.
- Do not use `pm.Constant` for parameters and do not use Gaussian processes for this task.
- For Student-T, give `nu` a continuous positive prior such as `pm.Exponential` or `pm.HalfNormal`.

Required Function Skeleton
- Define a function `def gen_model(observed_data)` that loads data via `observed_data["column_name"].values` (never a placeholder array literal).
- Inside `with pm.Model() as model:` declare the `MutableData` containers, the priors, and the likelihood `y_obs`.
- Sample with `trace = pm.sample(1000, tune=500, target_accept=0.90, chains=3, cores=1, random_seed=rng1, idata_kwargs={"log_likelihood": True})`.
- Then run `posterior_predictive = pm.sample_posterior_predictive(trace, random_seed=rng2, return_inferencedata=False)`.
- Return `(model, posterior_predictive, trace)` in that order.

Response Structure
- First, sketch a high-level probabilistic program for the data and explicitly cite which previous-round hypotheses (if any) the sketch addresses.
- Then write the full program in a Python code block, with comments on lines that address specific hypotheses.

Figure 13: Proposal prompt used by the BoxLM baseline for distribution fitting.

BoxLM Critic Prompt

You are a brilliant statistician specializing in critiquing and proposing revisions of probabilistic models. Your colleague has produced a list of PyMC programs that hypothesize `{model_class}` for the data; each program has been fit, plotted against the actual data, and scored with ELPD-LOO (higher is better). Your job is to figure out whether the proposed programs are consistent with the actual data by comparing the posterior predictive plots against the actual data, and to provide modeling-related guidance for the next round of revision.

Inputs
- Dataset description: `{dataset_description}`
- Column description: `{column_description}`
- Expert context: `{expert_context}`
- Programs from this round, each with its ELPD-LOO score and a posterior-predictive plot overlaid on the actual data.
- Hypotheses and synthesis from previous rounds (if any), which may be revised, removed, or extended.

Instructions
- Look at every posterior-predictive plot and explicitly describe the visual properties of the model versus the data.
- State natural-language hypotheses for any discrepancies between the proposed programs and the actual data, prioritizing the larger discrepancies.
- Using the programs and their ELPD-LOO scores, write a synthesis of which strategies and modeling choices did and did not work. The synthesis is passed back to your colleague, so make it as informative as possible. Restrict suggestions to modeling; do not suggest collecting more data.
- If hypotheses or a synthesis from previous rounds are provided, revise them based on the new evidence and briefly explain any additions or removals.

Output Format
Respond with two fenced blocks in this exact order:

````
``` Hypotheses
    * Hypothesis
    * Hypothesis
```

```Synthesis
    (Your synthesis here)
```
````

Domain-specific values
- For distribution fitting, `{model_class}` is *generative models*.
- For time series modeling, `{model_class}` is *Gaussian process models*.

Figure 14: Critic prompt used by the BoxLM baseline for both distribution fitting and time series modeling.

PyVision Agent Prompt

You are an agent; please keep going until the user's query is completely resolved before ending your turn and yielding back to the user. Only terminate your turn when you are sure that the problem is solved. Solve the following problem step by step. You now have the ability to selectively write executable Python code to enhance your reasoning process. The Python code will be executed by an external sandbox.

You MUST plan extensively before each function call, and reflect extensively on the outcomes of the previous function calls. Do not do this entire process by making function calls only, as this can impair your ability to solve the problem and think insightfully.

For all the provided images, in order, the *i*-th image has already been read into the global variable `image_clue_i` using `PIL.Image.open()`. When writing Python code, you can directly use these variables without needing to read them again.

Since you are dealing with the vision-related task, you MUST use the Python tool (e.g., the `matplotlib` library) to analyze or transform the image whenever it could improve your understanding or aid your reasoning. This includes but is not limited to zooming in, rotating, adjusting contrast, computing statistics, or isolating features. When you use `matplotlib` to visualize data or further process images, use `plt.show()` to display them; there is no need to save them. Do not use image-processing libraries like `cv2` or `PIL`. If you want to check the value of a variable, you MUST use `print()` to check it.

Code-execution Format
The output (wrapped in `<interpreter>output_str</interpreter>`) is returned to aid your reasoning. Each Python snippet must be a complete script including necessary imports, wrapped exactly as:

````
<code>
```python
code snippet
```
</code>
````

Image Information
`Image Width: {width}; Image Height: {height}`

User Query
`{query}`

Final Answer Format
The last part of your response must be in the following format:

```
<answer>
\boxed{'The final answer goes here.'}
</answer>
```

Domain-specific values
- For distribution fitting, `{query}` instructs the agent to propose and fit a PyMC probabilistic program for the empirical distribution shown in the histogram image, using the same PyMC coding guidelines as the VESTA agent.
- For time series modeling, `{query}` instructs the agent to propose and fit a PyMC Gaussian process program for the series shown in the time series plot, using the same PyMC coding guidelines as the VESTA agent.

Figure 15: Agent prompt used by the PyVision baseline for both distribution fitting and time series modeling.

Coding Agent Prompt for Time Series Modeling

You are solving a time-series problem. Infer the structural components that most likely generated the signal. The initial data plot is `initial_plot.png`.

Environment
- The dataset is `data.pkl`, a pickled pandas Series indexed by time. Read it directly with `pickle.load(open('data.pkl', 'rb'))`.
- You have access to Bash, Read, Write, Edit, Glob, and Grep. PyMC (`pm`), NumPy, pandas, SciPy, and matplotlib are installed.
- Run Python with `{python_path}` (the interpreter that has these packages).
- Everything you need is in this directory. All file access is confined here; attempts to read or write outside this directory will be blocked, so do not attempt to access the broader filesystem.

Use the `fit_pymc` MCP tool to fit and evaluate PyMC models against this data. When you call `fit_pymc`, you may pass a `distribution_family` argument (a list of the structural components you used) to label your proposal.

Code Contract for `fit_pymc`
- `data` is a pandas Series indexed by time, already in scope. Do not redefine it.
- Define a PyMC model in `with pm.Model() as model:`.
- Define `map_estimate = pm.find_MAP()`.
- Define `trend` as the predicted mean series over the observed times.

Figure 16: Coding-agent baseline prompts for time series.

Coding Agent Prompt for Distribution Fitting

You are solving a distribution-fitting problem. Infer the distribution family (or mixture) that most likely generated the data. The initial data plot is `initial_plot.png`.

Environment
- The dataset is `data.pkl`, a pickled numpy array. Read it directly with `pickle.load(open('data.pkl', 'rb'))`.
- You have access to Bash, Read, Write, Edit, Glob, and Grep. PyMC (`pm`), NumPy, SciPy, and matplotlib are installed.
- Run Python with `{python_path}` (the interpreter that has these packages).
- Everything you need is in this directory. All file access is confined here; attempts to read or write outside this directory will be blocked, so do not attempt to access the broader filesystem.

Use the `fit_pymc` MCP tool to fit and evaluate PyMC models against this data. When you call `fit_pymc`, you may pass a `distribution_family` argument (a list of distribution family names) to label your proposal.

Code Contract for `fit_pymc`
- `data` is a 1-D numpy array, already in scope. Do not redefine it.
- Define a PyMC model in `with pm.Model() as model:`.
- Define `map_estimate = pm.find_MAP()`.

Figure 17: Coding-agent baseline prompts for distribution fitting.