
000 AUTOSTAT: DSL-BASED AUTOMATED STATISTICAL 001 002 MODELING FROM NATURAL LANGUAGE 003 004

005 **Anonymous authors**

006 Paper under double-blind review

007 008 ABSTRACT 009

010 Statistical modeling plays a critical role and is widely used in data analysis across
011 diverse domains. Despite its importance, existing workflows remain cumbersome:
012 they rely on fragmented programming environments and domain-specific proba-
013 bilistic programming languages that are verbose and difficult to use, especially
014 for non-experts. Although many efforts have been made toward automated sta-
015 tistical modeling, the methods still suffer from low accuracy, high computational
016 cost, and heavy reliance on manual intervention. To address these challenges,
017 we present *AutoStat*, a novel Domain-Specific Language (DSL)-based framework
018 for automating statistical modeling. AutoStat leverages *StatModelDSL*, the first
019 compact and structured DSL that specifies complete modeling tasks in a unified
020 and portable form. AutoStat further enhances the automated process via interac-
021 tive modeling by integrating two agents – StatModelChatbot, which interactively
022 refines underspecified user requirements, and StatModelCopilot, which generates
023 executable DSL programs. With StatModelChatbot clarifying intent and StatMod-
024 elCopilot emitting executable DSL, AutoStat compiles and executes the specifi-
025 cation end-to-end, delivering the complex statistical models directly from natural-
026 language dialogue. We demonstrate that the proposed StatModelDSL affords both
027 LLM amenability and practical usability: when instantiated with GPT-4o, it yields
028 a **91.59%** reduction in error rate and a **5.89%** uplift in user preference over a Stan-
029 based workflow. Meanwhile, AutoStat achieves a **100%** syntax correctness rate
030 for DSL generation and a **98.76%** semantic passing rate, significantly surpassing
031 previous methods. Our dataset, codes, and models will be publicly released upon
032 acceptance.

033 1 INTRODUCTION

034 Statistical modeling (Box, 1976; Gelman et al., 1995) provides a principled framework for ana-
035 lyzing data by formalizing assumptions about the underlying data-generating process. It involves
036 constructing probabilistic models that link observed data with underlying variables, enabling both
037 interpretation and prediction. Statistical modeling is widely applied across domains such as eco-
038 nomics (Sims, 2012; Shavell, 2004), biology (Kaplan & Meier, 1958; Armitage & Doll, 2004), and
039 social sciences (Holland, 1986; Deegan Jr, 1979), where it helps researchers quantify uncertainty,
040 test hypotheses, and make predictions. However, existing workflows for statistical modeling remain
041 overly complex and unfriendly to users, as revealed by Gelman et al. (2020). On the one hand,
042 probabilistic programming languages (PPLs) such as Stan (Carpenter et al., 2017) and PyMC (Patil
043 et al., 2010) are syntactically verbose and often difficult to interpret. On the other hand, the overall
044 workflow is fragmented: even fitting a simple hierarchical regression typically requires preprocess-
045 ing data in a general-purpose language, writing dozens of lines of Stan code with explicit priors, and
046 then switching back for post-processing.

047 To simplify the workflow and improve usability, many efforts (Li et al., 2024; Gouk & Gao, 2024)
048 have been made to utilize the general knowledge from Large Language Model (LLM). Given a
049 task description, these methods typically prompt an LLM to synthesize end-to-end code in a cho-
050 sen PPL. Despite these attempts, we found that automated statistical modeling persistently suffers
051 from low accuracy, high computational cost, and heavy reliance on intervention due to three issues:
052 1) Insufficient specifications. Statistical models require fine-grained specifications (e.g., different
053 variable distributions and constraints), yet verbose PPL syntax often leads LLMs to misinterpret

054 critical details; for example, as shown in Appendix A, the number of sampling steps is frequently
055 misunderstood in the generated code, hindering the modeling process. 2) Fragmentation. Requiring
056 LLMs to bridge heterogeneous environments (*e.g.*, data preprocessing in Python while inference in
057 Stan (Carpenter et al., 2017)) often yields inconsistent outputs, resulting in multiple types of code
058 that are lengthy and difficult to read. (Figure 5 provides an example). 3) Lack of portability. Models
059 specified for one inference engine are not readily transferable across backends (*e.g.*, Stan (Carpenter
060 et al., 2017) to PyMC (Patil et al., 2010)), resulting in limited cross-engine comparability as well as
061 reduced validation capability. We identified that the root cause of the given limitations lies not only
062 in the modeling capability of LLMs but, more importantly, in the modeling-language substrate it-
063 self: Natural language is very underspecified for statistical modeling, while existing PPLs are overly
064 verbose, fragmented, and require substantial domain-specific expertise. Taken together, these issues
065 highlight the need for an intermediate abstraction that is structured and concise, providing a middle
066 ground between underspecified natural language and verbose PPLs.

067 Motivated by this, we present a novel automated statistical modeling framework, *AutoStat*. At
068 its core is *StatModelDSL*, a Domain-Specific Language (DSL) that represents complete statistical
069 modeling tasks in a unified and portable modeling language form. This abstraction offers three
070 main benefits: 1) Completeness and clarity: Every component of a task (data, parameters, model,
071 inference, output) is explicitly represented, reducing ambiguity and improving human readability; 2)
072 Unification and portability: A single DSL program can be compiled into multiple inference engines
073 (*e.g.*, Stan, PyMC), eliminating the need to switch between environments and avoiding framework
074 lock-in; and 3) LLM-friendliness: Its structured design makes fine-grained details explicit, enabling
075 LLMs to generate more accurate and reliable programs (Tam et al., 2024) than with free-form PPL
code. We validate these advantages in Section 4.5.

076 Since statistical modeling inherently requires fine-grained details that are difficult to specify in a
077 single attempt, we design *StatModelChatbot* to guide users through dialogue, progressively ground-
078 ing their intent into the structured components of *StatModelDSL* (*e.g.*, data, parameters, and model
079 blocks), enabling users to complete modeling tasks with ease. *AutoStat* incorporates two agents
080 to support the automation process (illustrated in Figure 1). *StatModelChatbot* interactively supple-
081 ments incomplete or ambiguous user descriptions to finalize all necessary task components, while
082 *StatModelCopilot* translates the clarified descriptions into executable *StatModelDSL* programs. To-
083 gether, these components allow users to specify and execute complex statistical models directly from
084 natural language, while ensuring that the resulting code remains precise, portable, and reliable.

085 We demonstrate that *StatModelDSL* affords both LLM amenability and practical usability: when
086 instantiated with an LLM, *e.g.*, GPT-4o, it yields a 91.59% reduction in error rate and a 5.89%
087 uplift in user preference over a Stan-based workflow. Building on this DSL, *AutoStat* attains a
088 100% syntax-correctness rate for DSL generation and a 98.76% semantic passing rate, significantly
089 surpassing prior methods. Our dataset, code, and models will be released upon acceptance.

090 In summary, our contributions include the following.
091

- 092 • We present *AutoStat*, an end-to-end DSL-based framework that converts natural language into
093 executable, reproducible statistical workflows. At its core is *StatModelDSL*—a concise, unified,
094 and standardized Domain-Specific Language that formalizes task specifications for automated
095 statistical modeling.
- 096 • We enable interactive statistical modeling that iteratively aligns users’ intent with *StatModelDSL*
097 through two agents that converse to capture and refine intent and then effectively produce the
098 specification into executable DSL programs—improving accuracy, usability, and efficiency.
- 099 • Extensive experiments on diverse statistical tasks show that the *StatModelDSL*-equipped *AutoStat*
100 delivers consistently higher accuracy, reliability, and reproducibility, outperforming the SOTA
101 automated modeling approaches.

102

2 RELATED WORK

103

2.1 STATISTICAL MODELING WORKFLOW

104 Current workflows for statistical modeling (Gelman et al., 2020) heavily rely on Probabilistic
105 Programming Languages (PPLs)(van de Meent et al., 2018; Krapu & Borsuk, 2019), such as

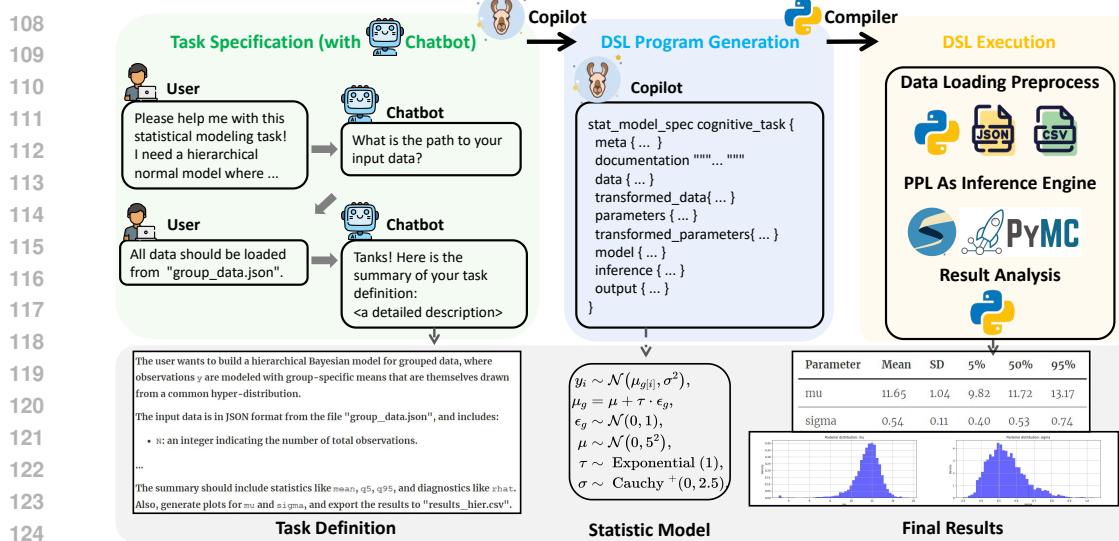


Figure 1: The workflow of AutoStat: StatModelChatbot first assists users in refining task details, after which StatModelCopilot generates the corresponding StatModelDSL program. The DSL compiler then executes this program to produce the final results.

Stan(Carpenter et al., 2017), PyMC (Patil et al., 2010), Turing.jl (Ge et al., 2018), and Pyro (Bingham et al., 2018). As noted by Gelman et al. (2020), while PPLs are powerful for statistical modeling, the complete workflow still requires external tools (e.g., Python or R) for data processing and analysis. Early attempts at automation, such as Fischer & Schumann (2003), transformed statistical models into automated data analysis pipelines, but the input rules were overly restrictive and the modeling expressiveness too limited for modern PPL environments. More recently, approaches leveraging LLMs (Gouk & Gao, 2024; Li et al., 2024) have emerged to translate natural language descriptions into PPL programs. However, these methods still suffer from fragmented environments and limited accuracy. In this work, we introduce a DSL-based framework that unifies the entire workflow. Leveraging LLMs, we make the workflow easy and reliable.

2.2 DOMAIN-SPECIFIC LANGUAGE FOR LARGE LANGUAGE MODELS

A Domain-Specific Language (DSL) (Fowler, 2010; Mernik et al., 2005) is a programming language tailored to a particular domain (e.g., Markdown, SQL). Its simplified and structured design makes task specification clearer for humans and more reliable for LLMs. DSLs can be broadly divided into two categories: 1) Standalone DSLs, which define their own syntax and compiler infrastructure. Examples include MoVer (Ma & Agrawala, 2025), a motion verification language for controllable motion generation, and SPCC (Li et al., 2025c), a specialized language for CAD modeling. 2) Embedded DSLs, which are implemented within a host language such as Python. Examples include Liang et al. (2022), who encode robot policies as Python programs for LLM-based policy control, Makatura et al. (2025), who design a DSL to capture metamaterials in a structured and expressive form, and Li et al. (2025a), who develop a Python-based DSL for material modeling and employ vision-language models for code generation. In this work, we propose StatModelDSL, a standalone DSL that models the entire statistical modeling workflow, and we use LLMs to automatically generate programs in this DSL.

3 METHOD

To address the verbosity of existing PPLs and enable a user-friendly, end-to-end automated statistical modeling workflow, we propose *AutoStat*. As illustrated in Figure 1, AutoStat leverages the concise and structured StatModelDSL (Section 3.1) to represent the entire statistical modeling task, replacing traditional PPLs. The StatModelChatbot (Section 3.2) interactively collects and clarifies task details from the user, after which the StatModelCopilot (Section 3.4) accurately generates specific DSL programs and executes them. With AutoStat, even novice users can complete complex statistical modeling tasks using only natural language.

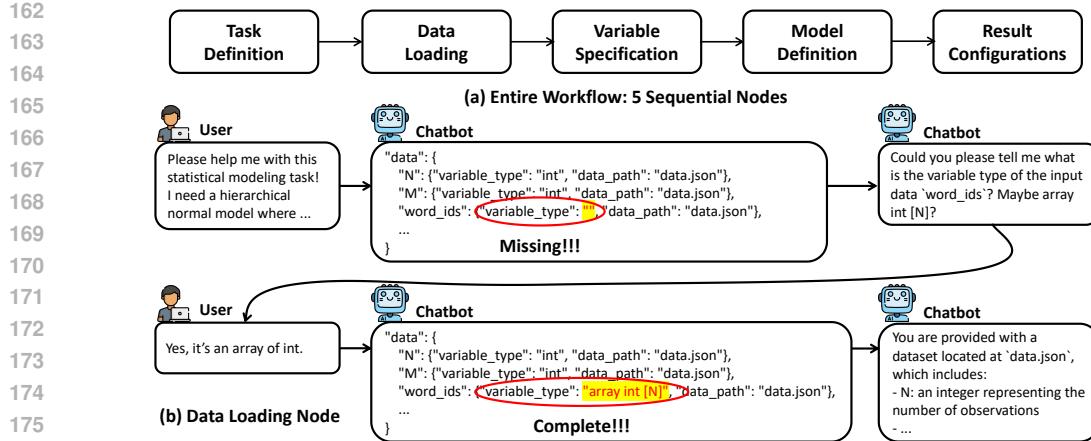


Figure 2: An illustration of the StatModelChatbot workflow. The chatbot leverages the user’s description to populate a predefined schema. If the provided information is insufficient to complete all required fields, the chatbot prompts the user for additional input until the schema is fully specified.

3.1 STATMODELDSL

We present StatModelDSL, a Domain-Specific Language that systematically expresses statistical modeling tasks in a structured and interpretable way. It integrates all components of the workflow, from data pre-processing and model specification to inference and result analysis, and can be compiled directly into executable programs across different probabilistic programming environments (*e.g.*, Stan, PyMC). The design of StatModelDSL emphasizes three key characteristics:

- **Clarity:** Each component of a statistical model is explicitly organized into separate blocks, ensuring that fine-grained details are represented in a transparent and structured manner.
- **Completeness:** The DSL captures the entire modeling pipeline, while our compiler automates execution end-to-end, reducing the fragmented and redundant steps of traditional workflows.
- **Portability:** The DSL can be translated into multiple inference engines, avoiding framework lock-in and enhancing flexibility. This cross-platform capability also facilitates the design of LLM-based agents, as they can generate a single unified DSL program without needing to handle environment-specific differences.

When executing the DSL, we first parse the code into an Abstract Syntax Tree (AST) using the Lark parser (Shinan, 2021), which captures the hierarchical structure of the program and facilitates traversal and manipulation. We then leverage a general-purpose programming language (Python) to handle data loading, preprocessing, and post-processing tasks such as plotting and result exporting, while the core statistical model is executed using the target PPL environment (*e.g.*, Stan or PyMC) to complete the end-to-end workflow. Appendix B provides a comprehensive description of StatModelDSL’s design, execution process, and illustrative examples, as well as demonstrations of its portability by compiling the program into different PPL backends.

3.2 STATMODELCHATBOT

Statistical modeling is inherently detail-intensive, requiring specifications such as variable types and distributional parameters. Since it is difficult for users to provide a complete description in a single attempt, we design a supportive chat agent (Wolf et al., 2019; Adiwardana et al., 2020), StatModelChatbot, that interactively assists users in refining task details, thereby ensuring completeness.

To ensure that the chatbot systematically verifies the presence of all necessary task information and produces outputs in a stable format, inspired by (Caufield et al., 2024; Lu et al., 2025; Shiri et al., 2024), we design a schema that specifies the key elements the chatbot must extract. Furthermore, to improve accuracy, following (Zhou et al., 2022), we decompose the task into five sequential nodes: task definition, data loading, variable specification, model definition, and result configuration.

As illustrated in Figure 2. For a certain node, the chatbot extracts key information from the user’s prompt to populate a pre-designed schema that specifies both required elements and optional ones. If

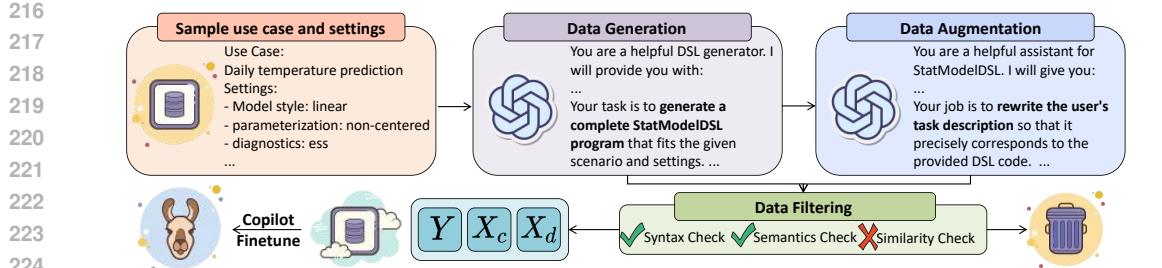


Figure 3: The pipeline of the StatModelDataset construction process, which contains three main steps: data generation, data augmentation, and data filtering.

all required fields are successfully filled, the node is considered complete, and the chatbot generates a natural-language summary for user confirmation before proceeding to the next node. If critical information is missing, the chatbot gives feedback to the user to provide the necessary details, and the process repeats. At the end of this interaction, we obtain a complete schema containing all essential specifications, along with a finalized natural-language task definition.

3.3 STATMODELDATASET

To effectively train and evaluate LLMs on our DSL, it is crucial to develop a diverse, and high-quality dataset. To this end, we construct StatModelDataset, whose construction pipeline, illustrated in Figure 3, comprises three stages: data generation, data augmentation, and data filtering.

3.3.1 DATA GENERATION

Due to the lack of existing datasets for statistical modeling, we leverage the strong in-context learning capabilities of LLMs (Brown et al., 2020; Yang et al., 2024; Li et al., 2025b) to generate synthetic data. To ensure the diversity of our dataset, we predefine 56 domains (*e.g.*, Time Series and Forecasting, Economics and Finance), each containing more than 10 potential use cases, resulting in a total of 590. In addition, we predefine a variety of model settings. For each use case, we randomly sample model settings and prompt GPT (Hurst et al., 2024) to generate both a concise task description X_c and the corresponding DSL code Y . To guarantee generation quality, we also provide LLMs with a natural-language description of the DSL syntax along with several detailed examples. Through this design, we are able to efficiently generate diverse base data.

3.3.2 DATA AUGMENTATION

A solely concise description is insufficient to capture the complexity of a statistical task; thus, we refine the initial task descriptions to provide greater detail. Specifically, we supply the DSL code to the LLM, along with illustrative examples, prompting it to generate a more comprehensive task description X_d . In this way, we construct a triplet dataset consisting of a concise task description X_c , a detailed task description X_d , and the corresponding DSL code Y . More details are illustrated in Appendix E.

3.3.3 DATA FILTERING

To ensure the quality of our dataset, the most critical step is filtering out erroneous and low-quality data. To this end, we designed the following data cleaning procedures tailored to our DSL.

- **Syntax check.** To ensure the syntax correctness of our DSL programs, we employ our DSL compiler to verify whether each DSL instance conforms to the syntax specification, while also checking for issues such as variable name reuse and other violations of syntactic rules.
- **Semantics check.** To assess whether each description is sufficiently complete and fully aligned with the DSL, we further prompt an LLM to verify the semantic consistency between them. This process ensures that the description does not omit critical details and that the DSL and the detailed description are matched at a fine-grained level. The prompt is shown in Appendix G.2
- **Similarity check.** To ensure dataset diversity and prevent the repetition of highly similar samples, we apply TF-IDF vectorization Salton et al. (1975) to all DSL code and remove instances that exceed a similarity threshold, thereby filtering out overly redundant code at the string level.

270 Although such strict filtering further reduces the dataset size, it ensures high-quality and reliable
271 data, thereby providing a solid foundation for the subsequent training of our StatModelCopilot.
272

273 For evaluation, the final test set (Table 7) contains 323 high-quality items, categorized into three
274 levels of complexity (simple, medium, and complex) based on the description X_d length. Both
275 training and testing datasets are verified by humans to ensure high quality.

276 3.4 STATMODELCOPILOT

279 After all task details are specified, an agent is responsible for converting the natural language de-
280 scription into a fully executable DSL program, enabling end-to-end automation. LLMs have demon-
281 strated strong capabilities in code generation (Chen et al., 2021; Jiang et al., 2024; Hui et al., 2024;
282 Guo et al., 2024), but for a new DSL that is both detail-intensive and instruction-heavy, relying
283 solely on in-context learning (Dong et al., 2024) is insufficient. To address this challenge, we train
284 our StatModelCopilot, which can follow complex statistical modeling instructions and generate syn-
285 tactically correct DSL programs that strictly adhere to the specifications.

286 To maximize the effectiveness of our dataset and ensure that the model both learns the syntax of
287 our DSL and faithfully follows complex user specifications, we adopt a curriculum learning ap-
288 proach (Bengio et al., 2009) with two training stages. In the first stage, we use concise task descrip-
289 tions as input prompts, with the corresponding DSL code as labels. This stage focuses on teaching
290 the LLM the syntax and structural rules of StatModelDSL, ensuring syntactic correctness. In the
291 second stage, we extend training to detailed task descriptions, where the model must capture and
292 follow all specified details, producing DSL programs that are not only syntactically valid but also
293 strictly aligned with the input requirements. We leverage the instruction tuning method to train our
294 models. The training losses are:

$$295 \mathcal{L}_1 = - \sum_{i=1}^{N_1} \log P(Y_i|(X_c)_i), \quad \mathcal{L}_2 = - \sum_{i=1}^{N_2} \log P(Y_i|(X_d)_i), \quad (1)$$

298 where N_1, N_2 denote the number of training samples for two stages, respectively.

300 4 EXPERIMENTS

303 We evaluate the proposed AutoStat framework to answer the following research questions:

- 304 • **RQ1:** How does AutoStat compared to other LLM-based methods?
- 305 • **RQ2:** What is the contribution to the designs in AutoStat (*i.e.*, two-stage training strategy, chatbot
306 assistance) to the performance?
- 307 • **RQ3:** How does the base LLM affect the performance of our AutoStat?
- 308 • **RQ4:** As the foundation of AutoStat, does StatModelDSL provide advantages over traditional
309 PPLs in LLM-based code generation and user usability?
- 310 • **RQ5:** Whether AutoStat can achieve strong performance and play a practical role in real-world
311 statistical modeling tasks.

314 All experiments are conducted on our test set mentioned in 3.3. During testing, the detailed task
315 description X_d is used as the input prompt, and the target StatModelDSL program Y serves as the
316 ground truth statistical modeling program.

318 4.1 EXPERIMENTAL SETTINGS

320 **Baselines.** Following prior work, we evaluate AutoStat using GPT-4o and GPT-4o-mini (Hurst
321 et al., 2024; Li et al., 2024), as well as Llama3-8B (Dubey et al., 2024; Gouk & Gao, 2024), under a
322 few-shot learning setup (Brown et al., 2020; Parnami & Lee, 2022), where models are given the DSL
323 specification and a few examples along with the task description. In contrast, our StatModelCopilot
324 directly generates the DSL program from the task description alone.

324 Table 1: Performance comparison between our AutoStat and GPT-based baselines. “*” indicates
 325 prompting with two in-context examples. The best performance is highlighted in **bold**.

327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377	Model	Syntax \uparrow				Semantics \uparrow			
		Simple	Medium	Complex	Avg.	Simple	Medium	Complex	Avg.
Llama3-8B	79.01	61.82	33.77	59.44	8.64	32.73	11.69	21.67	
Llama3-8B*	90.12	76.97	58.44	75.85	59.26	51.52	40.26	50.77	
GPT-4o-mini	98.77	96.36	88.31	95.05	90.12	82.42	48.05	76.16	
GPT-4o-mini*	98.77	97.58	93.51	96.90	93.83	93.94	79.22	90.40	
GPT-4o	97.53	99.39	96.10	98.14	92.59	93.33	77.92	89.47	
GPT-4o*	98.77	100.00	96.10	98.76	96.30	95.15	81.82	92.26	
AutoStat (1B)	98.77	97.58	92.21	96.59	91.36	90.30	68.83	85.45	
AutoStat (3B)	98.77	96.36	98.70	97.52	93.83	92.73	77.92	89.47	
AutoStat (8B)	100.00	100.00	100.00	100.00	98.77	99.39	97.40	98.76	

Metrics. To evaluate the accuracy of our automation pipeline, following (Chen et al., 2021; Kulal et al., 2019; Guo et al., 2025), we employ Pass@1 to assess the DSL code generation success rate. Specifically, we compare the generated DSL with the ground-truth DSL at both the AST level and after conversion to the target PPL code. A sample is considered successful only if all components, including model design, data processing, and variable specifications, exactly match the ground truth.

Implementation Details. We fine-tune Llama3-(1B, 3B, 8B) (Dubey et al., 2024) using the LoRA framework (Shen et al.) on a single NVIDIA A40 GPU. More details are shown in Table 6.

4.2 RESULTS ANALYSIS (RQ1)

Table 1 presents a comprehensive comparison between StatModelCopilot and all baseline methods. We summarize our key observations as follows:

- Our fine-tuned 8B Copilot achieves the best performance across tasks of varying difficulty, excelling at both the syntax and semantic levels. Specifically, it attains a **100% passing rate on syntax checks** and a **98.76% passing rate on semantic checks**. These results indicate that StatModelCopilot can accurately understand user requirements, capture nearly all critical task details, and generate programs that comply with our DSL specification. Compared to the weak performance of the pre-trained Llama baselines, the superior performance further validates the effectiveness of our training data and methodology.
- Providing additional examples significantly improves model performance. With just two examples, GPT-4o, GPT-4o-mini, and Llama3-8B all show substantial gains at the syntax and semantic levels. This demonstrates the strong in-context learning ability of powerful LLMs: supplementary examples help them better understand and apply our DSL, leading to more accurate task execution.
- Model capacity plays a crucial role in the performance of statistical modeling tasks, particularly in semantic understanding and handling complex tasks. In our complex-level semantic evaluations, GPT-4o outperforms GPT-4o-mini by nearly 30%. Similarly, our 8B StatModelCopilot achieves around 30% higher accuracy than its 1B counterpart and about 20% higher than the 3B version. These results suggest that for tasks requiring fine-grained semantic comprehension and precise specification, final performance strongly depends on the underlying model capacity, regardless of whether fine-tuning or prompt engineering is applied.

4.3 ABLATION STUDIES (RQ2)

To assess the effectiveness of our two-stage learning strategy and the StatModelChatbot, we perform ablation studies on the 8B StatModelCopilot: (1) “w/o Stage-One”: training only with detailed descriptions (X_d, Y); (2) “w/o Stage-Two”: training only with concise descriptions (X_c, Y); (3) “w/o Chatbot”: replacing the standardized prompts generated by StatModelChatbot with masked, user-tuned prompts X_d to simulate users’ prompts to examine its contribution to the overall workflow.

From the results shown in Table 2, we observe that:

378
379
Table 2: Effect of our training strategy and the StatModelChatbot.380
381
382
383
384
385
386

Model	Syntax ↑				Semantics ↑			
	Simple	Medium	Complex	Avg.	Simple	Medium	Complex	Avg.
Ours	100.00	100.00	100.00	100.00	98.77	99.39	97.40	98.76
- w/o Stage-One	98.77	100.00	96.10	98.76	96.30	95.15	81.82	92.26
- w/o Stage-Two	100.00	100.00	98.79	96.28	81.48	61.21	41.56	61.61
- w/o Chatbot	97.53	97.58	94.81	96.90	59.26	50.91	35.06	49.23

387
388
389
Table 3: Comparison across different base LLMs. For syntax and semantics, we report pass@1.
For speed, we report average seconds per batch (s/batch), and for memory, we report GPU VRAM
usage. The best results are highlighted in **bold**.390
391
392
393
394
395
396

Model	Size	Syntax ↑	Semantics ↑	Speed ↓	Memory ↓
Llama3	8B	100.00	98.76	19.72	41.22
Qwen3	8B	99.38	96.59	23.64	41.32
Qwen2.5-Coder	7B	98.76	94.43	19.22	41.64
Mistral	7B	99.69	92.26	27.44	41.52

397
398
399
400
401
402
• In the first stage, after training the LLM with concise task descriptions, the results show that the
model already achieves a strong syntax passing rate. This indicates that Stage-One successfully
enables the LLM to grasp the grammatical rules of our DSL, as expected. However, its relatively
weaker performance at the semantic level suggests that such training alone is insufficient for the
model to capture key information from task descriptions or to fully understand the user’s intent.
403
404
405
406
407
• Removing Stage-One training and directly learning from complex tasks affects performance, as
the model struggles to acquire the syntax and semantics of StatModelDSL from limited data. A
curriculum strategy—starting with simpler tasks and gradually increasing complexity—enables
the model to internalize DSL structures and capture semantic intent more effectively, consistent
with prior findings on curriculum learning (Bengio et al., 2009; Elman, 1993; Xu et al., 2020).
408
409
410
411
412
• Without the standard and complete prompt generated by StatModelChatbot, the performance of
our StatModelCopilot drops dramatically. On the one hand, user descriptions may be incom-
plete or underspecified, making it difficult for the Copilot to generate a DSL program that exactly
matches the target. On the other hand, while the training inputs X_d follow standardized expres-
sions, real user inputs are often more varied and irregular in format, which weakens the Copilot’s
performance. This mismatch is also reflected in the observed drop in syntax-level performance.
413414
415
4.4 EFFECT OF BASE LLMs (RQ3)416
417
418
419
To assess the impact of the underlying base model in StatModelCopilot, we conduct a comprehensive
comparison across LLMs of similar size. In addition to Llama3-8B (Dubey et al., 2024), we evaluate
Qwen3-8B (Yang et al., 2025), Qwen2.5-Coder-7B (Hui et al., 2024), and Mistral-7B (Jiang et al.,
2023), providing a systematic analysis of how different architectures affect performance.420
421
422
423
We evaluate both the syntax and semantics Pass@1 across the entire dataset. In addition, we assess
inference cost and efficiency. Specifically, we adopt vLLM (Kwon et al., 2023) with a fixed batch
size of 32, and measure the average inference time per batch and the GPU memory consumption.424
As shown in Table 3, we have the following observations:425
426
427
428
429
• Llama3 achieves the best overall performance both in terms of passing rate and memory usage.
• For models with similar sizes, the memory usage is roughly comparable; however, we observe
clear differences in inference speed. These differences primarily stem from architectural design
choices as well as the average generation length.
430
431
• We also test Qwen2.5-Coder to see if a code-specialized model would perform better, but it shows
no clear advantage. This may be because it lacks additional training on probabilistic programming
languages (Hui et al., 2024), which are central to our tasks.

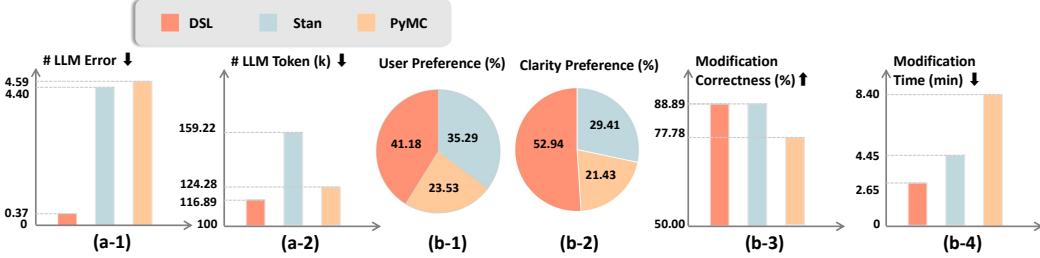


Figure 4: Performance comparison between the StatModelDSL-based workflow and PPL-based workflows. Subfigures (a-1) and (a-2) report experiments evaluating LLM-friendliness, while (b-1) to (b-4) present experiments evaluating user-friendliness.

4.5 STATMODELDSL EVALUATION (RQ4)

4.5.1 EXPERIMENTAL SETTINGS

We conduct both a quantitative evaluation and a user study to demonstrate that our StatModelDSL-based workflow is more accurate, efficient, and user-friendly compared to Stan and PyMC.

Quantitative evaluation. We uniformly prompt GPT-4o (Hurst et al., 2024) to generate programs on the test set under different environments. For fairness, when using StatModelDSL, we additionally provide its concise syntax specification to the model. We then evaluate the generated outputs along two dimensions: accuracy and cost. For accuracy, since implementations across different environments cannot be directly compared with rule-based checks, we adopt an LLM-as-a-judge approach (Liu et al., 2023; Zheng et al., 2023) (see Appendix G.2), where GPT-4o is asked to identify and count inconsistencies between the generated program and the task description. For cost, we measure the total number of tokens generated during inference on the entire dataset.

User study. To assess usability, we invite participants to modify programs under different environments to be familiar with different programming languages. Afterward, they are asked which environment they find most user-friendly, which they would prefer to use in the future, and which code representation they consider clearest and most readable. More details are shown in Appendix G.3.

4.5.2 RESULTS ANALYSIS

As shown in Figure 4, we have the following observations:

- As shown in (a-1) and (a-2), leveraging StatModelDSL dramatically improves the accuracy of GPT-4o: the error rate is reduced by more than **92.05%** compared to PPL-based workflows. In addition, token consumption is significantly lower, indicating that StatModelDSL is not only more accurate but also more efficient, making it a better fit for LLM-driven statistical modeling tasks.
- As shown in (b-1) and (b-2), most participants found our StatModelDSL programs clearer and more readable, and expressed a preference for using our DSL in similar tasks, demonstrating that StatModelDSL is genuinely user-friendly, benefiting from its clear and structured design.
- More specifically, results from (b-3) and (b-4) show that novice users achieved relatively high success rates when modifying both DSL and Stan programs. In addition, in terms of the time required for modification, DSL outperformed both Stan and PyMC by a clear margin, demonstrating that StatModelDSL is particularly user-friendly for novices, making code modification simpler and more efficient while also enhancing overall readability.

4.6 REAL-WORLD EVALUATION (RQ5)

To further evaluate the effectiveness of our AutoStat pipeline in real-world scenarios, we constructed two datasets of statistical modeling tasks:

- **Textbook-derived tasks.** We randomly curated 50 Bayesian modeling tasks from the official Stan Examples Repository¹, drawing exclusively from three authoritative textbooks in Bayesian statistics Gelman & Hill (2007); Lee & Wagenmakers (2014); Kéry & Schaub (2011).
- **Research paper-derived tasks.** To assess the utility of StatModelDSL in contemporary research settings, we analyzed 83 papers published in Bayesian Analysis² (2020–present). After automated

¹<https://github.com/stan-dev/example-models>

²<https://projecteuclid.org/journals/bayesian-analysis>

486 extraction and subsequent manual filtering, we selected 50 statistical modeling tasks with clear and
487 well-defined experimental descriptions.
488

489 All experimental settings follow exactly the same configuration as described in Section 4.5. We
490 compare the AutoStat pipeline against the traditional workflow based on Python and Stan, evaluating
491 their respective error rates and recording the syntax correctness rate of our DSL.
492

493 From the results shown in Table 4, we can observe that: 1) **Our AutoStat system is capable of handling**
494 **a wide range of real-world scenarios.** It can successfully complete
495 relatively straightforward data analysis tasks derived from textbooks, as
496 well as more complex statistical modeling tasks that appear in cutting-
497 edge research. 2) The formal task specification provided by StatModelDSL
498 **makes statistical modeling tasks clearer and more explicit**, re-

499 sulting in a lower error rate. Every detail of the task is fully represented, enabling our system
500 to perform more effectively, especially on complex modeling problems. Moreover, our StatModel-
501 Chatbot enables interactive communication with users, allowing the system to fully understand
502 task requirements. Even novice users with limited background in statistical modeling can easily use
503 the system to construct and execute sophisticated statistical experiments.
504

505 For these evaluations, we will release all testing data, task descriptions, DSL implementations, and
506 detailed metadata—including the source papers and the exact locations of the corresponding experiments
507 within those papers—together with our full system. Additional examples and further details
508 can be found in the Appendix H.
509

510

511

512 5 CONCLUSION AND FUTURE WORK

513

514

515

516

517

518

In this work, we propose AutoStat, a unified framework for automating statistical modeling. AutoStat leverages StatModelDSL, the first Domain-Specific Language designed to simplify the entire workflow of statistical modeling. Building on this foundation, AutoStat also consists of StatModelChatbot and StatModelCopilot, which enable end-to-end automation of the task. With our framework, even novice users can complete complex statistical modeling tasks simply by delivering the complex statistical models directly from natural-language dialogue. Through extensive experiments, we demonstrate that our method improves both accuracy and usability.

519

520

521

522

523

524

Looking forward, our framework offers potential for further scaling and improvement. StatModelDSL can be extended to support richer data structures, more flexible data input functions, and advanced output processing. The StatModelCopilot can be further enhanced by incorporating real-world data for both training and evaluation, ensuring more reliable performance. Additionally, developing a more lightweight version of the model (*e.g.*, using MoE-based architectures) would improve efficiency and deployability for users. We also envision an online platform that allows users to access the full workflow without requiring local GPU resources, making automated statistical modeling more widely accessible.

525

526

527

528 535 REPRODUCIBILITY STATEMENT

536

537

538

539

Our code, dataset, and model weights will be publicly released upon acceptance. All implementation details and training cost are shown in Appendix F Table 6. Our dataset, codes, and models will be publicly released upon acceptance.

Table 4: Comparison of error rates between AutoStat and the baseline. “Syntax” refers to the passing rate of syntax checks performed by our DSL parser, and “Error” denotes the number of inconsistencies between the generated output and the task requirements as judged by the LLM.

	Textbook		Research paper	
	Syntax (%)	# Error	Syntax (%)	# Error
Python + Stan	-	0.00	-	8.90
AutoStat	100	0.00	100	5.48

540 REFERENCES

541

542 Daniel Adiwardana, Minh-Thang Luong, David R So, Jamie Hall, Noah Fiedel, Romal Thoppilan,
543 Zi Yang, Apoorv Kulshreshtha, Gaurav Nemade, Yifeng Lu, et al. Towards a human-like open-
544 domain chatbot. *arXiv preprint arXiv:2001.09977*, 2020.

545 Peter Armitage and Richard Doll. The age distribution of cancer and a multi-stage theory of car-
546 cinogenesis. *British journal of cancer*, 91(12):1983–1989, 2004.

547

548 Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In
549 *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

550 Mario Beraha, Riccardo Corradin, et al. Bayesian nonparametric model-based clustering with in-
551 tractable distributions: An abc approach. *Bayesian Analysis*, 1(1):1–28, 2024.

552

553 Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis
554 Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep
555 universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2018. URL <https://api.semanticscholar.org/CorpusID:53038373>.

556

557 George EP Box. Science and statistics. *Journal of the American Statistical Association*, 71(356):
558 791–799, 1976.

559

560 Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal,
561 Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are
562 few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

563

564 Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betan-
565 court, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic program-
566 ming language. *Journal of statistical software*, 76:1–32, 2017.

567

568 J Harry Caufield, Harshad Hegde, Vincent Emonet, Nomi L Harris, Marcin P Joachimiak, Nicolas
569 Matentzoglu, HyeongSik Kim, Sierra Moxon, Justin T Reese, Melissa A Haendel, et al. Struc-
570 tured prompt interrogation and recursive extraction of semantics (spires): A method for populating
571 knowledge bases using zero-shot learning. *Bioinformatics*, 40(3):btae104, 2024.

572

573 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared
574 Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large
575 language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

576

577 John Deegan Jr. Constructing statistical models of social processes. *Quality & Quantity*, 13(2),
578 1979.

579

580 Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu,
581 Zhiyong Wu, Baobao Chang, et al. A survey on in-context learning. In *Proceedings of the 2024*
582 *Conference on Empirical Methods in Natural Language Processing*, pp. 1107–1128, 2024.

583

584 Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha
585 Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models.
586 *arXiv e-prints*, pp. arXiv–2407, 2024.

587

588 Jeffrey L Elman. Learning and development in neural networks: The importance of starting small.
589 *Cognition*, 48(1):71–99, 1993.

590

591 Bernd Fischer and Johann Schumann. Autobayes: a system for generating data analysis programs
592 from statistical models. *J. Funct. Program.*, 13(3):483–508, May 2003. ISSN 0956-7968. doi: 10.
593 1017/S0956796802004562. URL <https://doi.org/10.1017/S0956796802004562>.

594

595 Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

596

597 Hong Ge, Kai Xu, and Zoubin Ghahramani. Turing: a language for flexible probabilistic inference.
598 In *International conference on artificial intelligence and statistics*, pp. 1682–1690. PMLR, 2018.

599

600 Andrew Gelman and Jennifer Hill. *Data analysis using regression and multilevel/hierarchical mod-
601 els*. Cambridge university press, 2007.

594 Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*. Chapman
595 and Hall/CRC, 1995.
596

597 Andrew Gelman, Aki Vehtari, Daniel Simpson, Charles C Margossian, Bob Carpenter, Yuling Yao,
598 Lauren Kennedy, Jonah Gabry, Paul-Christian Bürkner, and Martin Modrák. Bayesian workflow.
599 *arXiv:2011.01808*, 2020.

600 Henry Gouk and Boyan Gao. Automated prior elicitation from large language models for bayesian
601 logistic regression. In *The 3rd International Conference on Automated Machine Learning*, 2024.
602

603 Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao
604 Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–
605 the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

606 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,
607 Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms
608 via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

609

610 Paul W. Holland. Statistics and causal inference. *Journal of the American Statistical Association*, 81(396):945–960, 1986. ISSN 01621459, 1537274X. URL <http://www.jstor.org/stable/2289064>.

611

612

613 Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang,
614 Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*,
615 2024.

616

617 Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Os-
618 trow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv:2410.21276*,
619 2024.

620

621 Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chap-
622 lot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier,
623 Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril,
624 Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL <https://arxiv.org/abs/2310.06825>.

625

626 Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. A survey on large language
627 models for code generation. *arXiv preprint arXiv:2406.00515*, 2024.

628

629 E. L. Kaplan and Paul Meier. Nonparametric estimation from incomplete observations. *Journal of
630 the American Statistical Association*, 53(282):457–481, 1958. ISSN 01621459, 1537274X. URL
631 <http://www.jstor.org/stable/2281868>.

632

633 Marc Kéry and Michael Schaub. *Bayesian population analysis using WinBUGS: a hierarchical
634 perspective*. Academic press, 2011.

635

636 Christopher Krapu and Mark E. Borsuk. Probabilistic programming: A review for envi-
637 ronmental modellers. *Environ. Model. Softw.*, 114:40–48, 2019. URL <https://api.semanticscholar.org/CorpusID:67880715>.

638

639 Sumith Kulal, Panupong Pasupat, Kartik Chandra, Mina Lee, Oded Padon, Alex Aiken, and Percy S
640 Liang. Spoc: Search-based pseudocode to code. *Advances in Neural Information Processing
641 Systems*, 32, 2019.

642

643 Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E.
644 Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model
645 serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating
646 Systems Principles*, 2023.

647 Michael D Lee and Eric-Jan Wagenmakers. *Bayesian cognitive modeling: A practical course*. Cam-
bridge university press, 2014.

648 Beichen Li, Rundi Wu, Armando Solar-Lezama, Changxi Zheng, Liang Shi, Bernd Bickel, and Wo-
649 jciech Matusik. Vlmaterial: Procedural material generation with large vision-language models.
650 *arXiv preprint arXiv:2501.18623*, 2025a.
651

652 Jia Li, Chongyang Tao, Jia Li, Ge Li, Zhi Jin, Huangzhao Zhang, Zheng Fang, and Fang Liu. Large
653 language model-aware in-context learning for code generation. *ACM Transactions on Software
654 Engineering and Methodology*, 34(7):1–33, 2025b.
655

656 Jiahao Li, Weijian Ma, Xueyang Li, Yunzhong Lou, Guichun Zhou, and Xiangdong Zhou. Cad-
657 llama: leveraging large language models for computer-aided design parametric 3d model gener-
658 ation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 18563–
659 18573, 2025c.
660

661 Michael Y Li, Emily B Fox, and Noah D Goodman. Automated statistical model discovery with
language models. *arXiv:2402.17879*, 2024.
662

663 Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and
Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint
arXiv:2209.07753*, 2022.
664

665 Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg
666 evaluation using gpt-4 with better human alignment. *arXiv preprint arXiv:2303.16634*, 2023.
667

668 Yaxi Lu, Haolun Li, Xin Cong, Zhong Zhang, Yesai Wu, Yankai Lin, Zhiyuan Liu, Fangming Liu,
669 and Maosong Sun. Learning to generate structured output with schema reinforcement learning.
670 *arXiv preprint arXiv:2502.18878*, 2025.
671

672 Jiaju Ma and Maneesh Agrawala. Mover: Motion verification for motion graphics animations. *ACM
Transactions on Graphics (TOG)*, 44(4):1–17, 2025.
673

674 Liane Makatura, Benjamin Jones, Siyuan Bian, and Wojciech Matusik. Metagen: A dsl, database,
675 and benchmark for vlm-assisted metamaterial generation. *arXiv preprint arXiv:2508.17568*,
676 2025.
677

678 Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific
languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.
679

680 Archit Parnami and Minwoo Lee. Learning from few examples: A summary of approaches to few-
681 shot learning. *arXiv preprint arXiv:2203.04291*, 2022.
682

683 Anand Patil, David Huard, and Christopher J Fonnesbeck. Pymc: Bayesian stochastic modelling in
python. *Journal of statistical software*, 35:1–81, 2010.
684

685 Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing.
686 *Communications of the ACM*, 18:613–620, 1975.
687

688 Steven Shavell. *Foundations of economic analysis of law*. Harvard University Press, 2004.
689

690 Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, et al. Lora: Low-rank
adaptation of large language models.
691

692 Erez Shinan. Lark., 2021. URL <https://github.com/lark-parser/lark>.
693

694 Fatemeh Shiri, Farhad Moghimifar, Reza Haffari, Yuan-Fang Li, Van Nguyen, and John Yoo. De-
695 compose, enrich, and extract! schema-aware event extraction using llms. In *2024 27th Interna-
tional Conference on Information Fusion (FUSION)*, pp. 1–8. IEEE, 2024.
696

697 Christopher A. Sims. Statistical modeling of monetary policy and its effects. *American Economic
Review*, 102(4):1187–1205, June 2012. doi: 10.1257/aer.102.4.1187. URL <https://www.aeaweb.org/articles?id=10.1257/aer.102.4.1187>.
698

699 Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen.
700 Let me speak freely? a study on the impact of format restrictions on performance of large language
701 models. *arXiv preprint arXiv:2408.02442*, 2024.

702 Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduc-
703 tion to probabilistic programming. *ArXiv*, abs/1809.10756, 2018. URL <https://api.semanticscholar.org/CorpusID:52893693>.
704

705 Thomas Wolf, Victor Sanh, Julien Chaumond, and Clement Delangue. Transfertransfo: A
706 transfer learning approach for neural network based conversational agents. *arXiv preprint*
707 *arXiv:1901.08149*, 2019.
708

709 Benfeng Xu, Licheng Zhang, Zhendong Mao, Quan Wang, Hongtao Xie, and Yongdong Zhang.
710 Curriculum learning for natural language understanding. In *Proceedings of the 58th annual meet-
711 ing of the association for computational linguistics*, pp. 6095–6104, 2020.
712

713 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu,
714 Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint*
715 *arXiv:2505.09388*, 2025.
716

717 Tong Yang, Yu Huang, Yingbin Liang, and Yuejie Chi. In-context learning with representations:
718 Contextual generalization of trained transformers. *Advances in Neural Information Processing
Systems*, 37:85867–85898, 2024.
719

720 Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang,
721 Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and
722 chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.
723

724 Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyuan Luo, Zhangchi Feng, and
725 Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Pro-
726 ceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume
727 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguis-
728 tics. URL <http://arxiv.org/abs/2403.13372>.
729

730 Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuur-
731 mans, Claire Cui, Olivier Bousquet, Quoc Le, et al. Least-to-most prompting enables complex
732 reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.
733

734

735

736

737

738

739

740

741

742

743

744

745

746

747

748

749

750

751

752

753

754

755

756	USE OF LLMs IN OUR WORK	
757		
758	In this paper, we leverage LLMs in several ways: (1) polishing the writing of our manuscript, (2)	
759	retrieving relevant papers, (3) refining the design of prompts, and (4) generating part of our dataset	
760	(as described in Section 3.3). Importantly, all outputs produced by LLMs were carefully reviewed	
761	and verified by humans to ensure accuracy and reliability.	
762		
763	CONTENTS	
764		
765		
766	1 Introduction	1
767		
768	2 Related Work	2
769	2.1 Statistical Modeling Workflow	2
770	2.2 Domain-Specific Language for Large Language Models	3
771		
772		
773	3 Method	3
774	3.1 StatModelDSL	4
775	3.2 StatModelChatbot	4
776	3.3 StatModelDataset	5
777	3.3.1 Data Generation	5
778	3.3.2 Data Augmentation	5
779	3.3.3 Data Filtering	5
780	3.4 StatModelCopilot	6
781		
782		
783		
784		
785	4 Experiments	6
786	4.1 Experimental Settings	6
787	4.2 Results Analysis (RQ1)	7
788	4.3 Ablation Studies (RQ2)	7
789	4.4 Effect of Base LLMs (RQ3)	8
790	4.5 StatModelDSL Evaluation (RQ4)	9
791	4.5.1 Experimental Settings	9
792	4.5.2 Results analysis	9
793	4.6 Real-World Evaluation (RQ5)	9
794		
795		
796		
797		
798	5 Conclusion and Future Work	10
799		
800		
801	A Environments Comparison: An Example	16
802		
803	B StatModelDSL	17
804	B.1 DSL Components	17
805	B.2 DSL Execution	18
806	B.3 Example: Linear Regression in StatModelDSL	20
807		
808		
809		
	C StatModelChatbot	22

810	C.1 Extraction Schema	22
811	C.2 StatModelChatbot Workflow	22
812		
813		
814	D An entire example using AutoStat, from task specification to DSL	25
815		
816	E StatModelDataset	28
817		
818	F StatModelCopilot	30
819		
820		
821	G More Experimental Details	30
822	G.1 Details about our test dataset	30
823		
824	G.2 LLM-as-a-judge	30
825		
826	G.3 User Study	31
827		
828	H Real-world Evaluation	31
829	H.1 Task Collections	31
830		
831	H.2 Examples	32
832		

A ENVIRONMENTS COMPARISON: AN EXAMPLE

```

Stan
import json
import pandas as pd
import numpy as np
import az
import azplotlib.pyplot as plt

# Load the data
with open('benchmark/data/data20.json') as f:
    data = json.load(f)

x = data['x']
y = np.array(data['y'])
year = np.array(data['year'])

# Compute y_difff
y_difff = np.diff(y)

# Stan model code
stan_code = """
data {
    int N;
    int N_difff;
    vector[N_difff] y_difff;
}
parameters {
    real mu_latent;
    real year<0> sigma_latent;
    vector[N_difff] latent_effect;
}
model;
    mu_latent ~ normal(0, 3);
    sigma_latent ~ cauchy(0, 3.5);
    latent_effect ~ normal(0, 3);
}
for (i in 1:N_difff) {
    y_difff[i] ~ normal(mu_latent + latent_effect[i], sigma_latent);
}
"""

# Prepare data for Stan
stan_data = {
    "N_difff": N - 1,
    "y_difff": y_difff
}

# Compile the stan model
sm = pystan.StanModel(model_code=stan_code)

# Fit the model using Stan
fit = sm.sample(data=stan_data, iter=3000, warmup=1000, chains=4,
                 control={"adapt_delta": 0.95})

# Extract results
results = fit.extract(permuted=True)
mu_latent_samples = results['mu_latent']
sigma_latent_samples = results['sigma_latent']
latent_effect_samples = results['latent_effect']

# Extract 95th percentile
q95_mu_latent = np.percentile(mu_latent_samples, 95)
q95_sigma_latent = np.percentile(sigma_latent_samples, 95)

# Save results to CSV
results_df = pd.DataFrame({
    'mu_latent': mu_latent_samples,
    'sigma_latent': sigma_latent_samples,
    'q95_mu_latent': q95_mu_latent,
    'q95_sigma_latent': q95_sigma_latent
})
q95_mu_latent = np.percentile(mu_latent_samples, 95th_percentile)

# Check for divergent transitions
divergences = trace.sample_stats.diverging.sum()
print(f"Number of divergent transitions: {divergences}")

# Export results to CSV
results = {
    'mu_latent': [mu_latent, 'sigma_latent'] = [f'latent_effect[{i}]' for i in range(N-1)],
    'q95_mu_latent': [q95_mu_latent, 'sigma_latent'] = [f'latent_effect[{i}]' for i in range(N-1)],
    '95th_percentile': [95th_percentile, 'sigma_latent'] = [f'latent_effect[{i}]' for i in range(N-1)],
}
df = pd.DataFrame(results)
df.to_csv('results_patient_readmission.csv', index=False)

PyMC
import pymc as pm
import numpy as np
import pandas as pd
import az
import azplotlib.pyplot as plt
import json

# Load the data
with open('benchmark/data/data20.json', 'r') as f:
    data = json.load(f)

N = data['N']
y = np.array(data['y'])
year = np.array(data['year'])

# Compute y_difff
y_difff = np.diff(y)

# Define the model
with pm.Model() as model:
    # Priors
    mu_latent = pm.Normal('mu_latent', mu=0, sigma=3)
    sigma_latent = pm.HalfCauchy('sigma_latent', beta=2.5)

    # Latent effects
    latent_effect = pm.Normal('latent_effect', mu=0, sigma=3, shape=N-1)

    # Likelihood
    y_obs = pm.Normal('y_obs', mu=mu_latent + latent_effect, sigma=sigma_latent,
                      observed=y_difff)

    # Inference
    trace = pm.sample(2000, tune=1000, chains=4, target_accept=0.95,
                      return_inferencedata=True)

    # Posterior summary
    summary = az.summary(trace, MLE_prob0.95)
    summary.to_csv('results_patient_readmission.csv')

    # Diagnostics for divergent transitions
    divergences = az.plot_trace(trace, var_names=['mu_latent', 'sigma_latent'],
                                compact=True)
    plt.show()

    # Plotting posterior distributions
    az.plot_posterior(trace, var_names=['mu_latent', 'sigma_latent'])
    plt.show()

StatModelDSL
stat_model_spec_patient_readmission_latent {
    stat_id: 1
    author: "StatModel"
    dsl_version: "1.0";
    model_version: "1.0";
    model_language: "Stan";
    target_language: "Stan";
}

documentation """
This model estimates patient readmission rates using a latent variable approach. It incorporates informative priors and focuses on uncertainty quantification.
"""

data {
    source_n: int! From json(path='benchmark/data/data20.json');
    source_y: vector[N] From json(path='benchmark/data/data20.json');
    source_year: array[N] int From json(path='benchmark/data/data20.json');
}

transformed_data {
    y_difff: vector[N-1];
    for (i in 1:(N-1)) {
        y_difff[i] = y[i+1] - y[i];
    }
}

parameters {
    mu_latent: real ~ normal(0, 3);
    sigma_latent: real where sigma_latent > 0 ~ cauchy(0, 2.5);
    latent_effect: vector[N-1] ~ normal(0, 3);
}

model {
    for (i in 1:(N-1)) {
        y_difff[i] ~ normal(mu_latent + latent_effect[i], sigma_latent);
    }
    for (i in 1:(N-1)) {
        y_difff[i] ~ normal(mu_latent + latent_effect[i], sigma_latent);
    }
}

inference {
    method: mcmc;
    seed: 42;
    chains: 4;
    num_samples: 2000;
    num_warmup: 1000;
}

output {
    monitor: [mu_latent, sigma_latent, latent_effect];
    summary_stats: [95th_percentile];
    divergences: [divergent_transitions];
    plots: [mu_latent, sigma_latent];
    export_results_to: "results_patient_readmission.csv";
}

```

Figure 5: We prompt GPT-4o to generate code from the same task description under different probabilistic programming environments.

As illustrated in Figure 5, we instructed GPT-4o to generate code under different probabilistic programming environments using the identical detailed task description. Through this experiment, three key observations can be explicitly drawn:

864 • The Stan-generated code is notably lengthy, and there exists a stark discrepancy in coding style
 865 between Stan and Python, leading to weak readability. Additionally, the overall implementation
 866 is highly complex and cumbersome.
 867 • While the PyMC code is relatively concise, it suffers from poor clarity and poses a steep learning
 868 curve for beginners. For instance, in this task, we require the “`sigma_latent`” variable to follow
 869 a Cauchy distribution with the constraint of being greater than 0. In PyMC, this necessitates the
 870 specific use of “`HalfCauchy`”—an operation that beginners may struggle to implement straight-
 871 forwardly.
 872 • In contrast, our proposed DSL achieves both conciseness and clarity. It streamlines numerous
 873 repetitive yet essential procedures (*e.g.*, plotting and data loading) into single, unambiguous lines
 874 of code. Furthermore, the statistical model component of our DSL is remarkably intuitive: for the
 875 “`sigma_latent`” variable mentioned above, we simply apply the constraint “`> 0`” for modification.
 876 This approach also affords greater flexibility to the model formulation.

877 We further instructed GPT-4o to act as a critic, tasked with identifying all discrepancies between the
 878 generated code and the task description. The key findings are as follows:
 879

880 • Regarding the Stan code, there is an error in the sampling iterations. The task description explicitly
 881 requires sampling 2000 iterations with 1000 warmup iterations, yet the generated code fails to
 882 adhere to this specification.
 883 • For the PyMC code, a mistake is present in the results saving process. As specified in the task, the
 884 target metric to be saved is `q95`, but the generated code incorrectly saves `hdi_97.5%` instead.
 885 • In contrast, the code generated using our DSL is entirely accurate. GPT-4o did not detect any
 886 mismatches between the DSL code and the task definition.

887 B STATMODELDSL

888 B.1 DSL COMPONENTS

889 Our proposed StatModelDSL consists of several building blocks, each serving a specific role in
 890 statistical modeling. The overall structure is as follows:
 891

```
892
893
894
895   stat_model_spec <model_name> {
896     meta { ... }
897     documentation "..." | """..."""
898     data { ... }
899     transformed_data { ... } (Optional)
900     parameters { ... }
901     transformed_parameters { ... } (Optional)
902     model { ... }
903     inference { ... }
904     output { ... }
905   }
```

906 Block Descriptions:

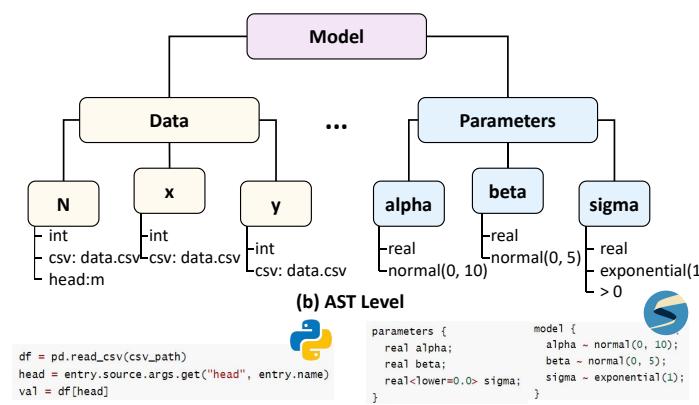
907 • **meta**: Specifies metadata such as inference engine, author, and version information.
 908 • **documentation**: Provides human-readable notes or descriptions of the program in natural lan-
 909 guage.
 910 • **data**: Defines input data sources and corresponding constraints.
 911 • **transformed_data**: (Optional) Declares new variables derived from input data.
 912 • **parameters**: Lists model parameters with constraints and prior distributions.
 913 • **transformed_parameters**: (Optional) Defines transformed parameters as variables for down-
 914 stream modeling.
 915 • **model**: Contains the core statistical model specification.
 916 • **inference**: Configures inference algorithms and settings.
 917 • **output**: Specifies outputs, including monitored parameters, summaries, diagnostics, plots, and
 918 export options.

```

918 stat_model_spec linear_regression_example {
919 ...
920 ...
921 data {
922   source N: int from csv(
923     path="data.csv", head="num");
924   source x: vector from csv(path="data.csv");
925   source y: vector[N] from csv(path="data.csv");
926 }
927 ...
928 ...
929 }
930

```

(a) DSL Program



(b) AST Level

(c) Python Process + PPL Inference

Figure 6: Our StatModelDSL execution pipeline.

B.2 DSL EXECUTION

As illustrated in Figure 6, our StatModelCompiler takes two steps to execute a DSL program.

- **(a) DSL → (b) AST.** For a DSL program, we first parse the code using the Lark parser, converting it into a hierarchical tree structure with the format “program → blocks → entries”. This tree representation facilitates program comprehension and allows us to read, validate, and process the code in a structured, block-wise manner.
- **(b) AST → (c) Python+PPL.** At the AST level, we handle each block differently. For example, the data block is used to load and validate input data according to its specifications, while the parameters block is converted into the corresponding code in the target PPL.

This is the execution EBNF of StatModelDSL, which formally specifies the rules for parsing and compiling the entire modeling workflow.

```

948 Execution EBNF of StatModelDSL
949
950 start: stat_model_spec
951
952 stat_model_spec: "stat_model_spec" NAME "(" meta_block documentation_block? data_block
953 transformed_data_block? parameters_block transformed_parameters_block? model_block
954 inference_block? output_block? ")"
955
956 meta_block: "meta" "(" meta_pair* ")"
957 meta_pair: NAME ":" value ";"
958
959 documentation_block: "documentation" (MULTILINE_STRING_LITERAL | ESCAPED_STRING)
960
961 data_block: "data" "(" data_decl* ")"
962 data_decl: "source" NAME ":" type (constraint)? "from" source_type "(" arg_list? ")" ";"
963
964 trans_data_decl: NAME ":" type (constraint)? ";"
965
966 constraint: "where" bool_expr
967
968 ?bool_expr: bool_expr "and" bool_expr -> and_
969   | bool_expr "or" bool_expr -> or_
970   | "not" bool_expr -> not_
971   | expr
972   | "(" bool_expr ")"
973
974 source_type: NAME
975
976 transformed_data_block: "transformed_data" "(" (trans_data_decl | assign_stmt |
977 compound_assign_stmt | for_stmt | if_stmt)* ")"
978
979 parameters_block: "parameters" "(" param_decl* ")"
980 param_decl: NAME ":" type constraint? prior? ";"
981 prior: "~" distribution
982
983 transformed_parameters_block: "transformed_parameters" "(" (trans_data_decl | assign_stmt |

```

```

972     compound_assign_stmt | for_stmt | if_stmt)* }"
973
974     model_block: "model" "{" stmt_body* "}"
975
976     distribution: NAME "(" [expr (",", expr)*] ")"
977
978     inference_block: "inference" "(" "method" ":" NAME ";" settings_block? ")"
979     settings_block: "settings" ":" "(" inference_setting* ")"
980     inference_setting: NAME ":" value ";"
981
982     output_block: "output" "(" output_stmt* ")"
983     output_stmt: NAME ":" value_or_list ";"
984
985     arg_list: arg (",", arg)*
986     arg: NAME "=" value
987
988     value_or_list: value | "[" [value (",", value)*] "]"
989     value: NUMBER | BOOLEAN | ESCAPED_STRING | NAME
990
991     for_stmt: "for" "(" NAME "in" expr ":" expr ")" "{" stmt_body* "}"
992     if_stmt: "if" "(" bool_expr ")" "(" then_body ")" ("else" "(" else_body ")" )?
993     then_body: stmt_body*
994     else_body: stmt_body*
995     assign_stmt: assign_target "=" expr ";"
996     compound_assign_stmt: assign_target (PLUS_EQ | MINUS_EQ | STAR_EQ | DIV_EQ) expr ";"
997     assign_target: NAME | indexed_access
998
999     dist_stmt: assign_target "~~" distribution ";"
1000
1001     ?stmt_body: assign_stmt | compound_assign_stmt | dist_stmt | for_stmt | trans_data_decl | if_stmt
1002
1003     ?expr: expr GT term    -> gt
1004         | expr LT term    -> lt
1005         | expr GTE term   -> gte
1006         | expr LTE term   -> lte
1007         | expr EQ term    -> eq
1008         | expr NEQ term   -> neq
1009         | expr PLUS term  -> add
1010         | expr MINUS term -> sub
1011         | term
1012
1013     ?term: term STAR factor -> mul
1014         | term DIV factor -> div
1015         | term POW factor -> pow
1016         | factor
1017
1018     ?factor: function_call
1019         | log_prob_call
1020         | NAME
1021         | indexed_access
1022         | NUMBER
1023         | ESCAPED_STRING
1024         | BOOLEAN
1025         | "(" expr ")"
1026
1027     function_call: NAME "(" [expr (",", expr)*] ")"
1028     log_prob_call: NAME "(" expr ("|" expr (",", expr)*) ")"
1029
1030     PLUS: "+"
1031     MINUS: "-"
1032     STAR: "*"
1033     DIV: "/"
1034     POW: "^"
1035     GT: ">"
1036     LT: "<"
1037     GTE: ">="
1038     LTE: "<="
1039     EQ: "==""
1040     NEQ: "!="
1041     PLUS_EQ: "+="
1042     MINUS_EQ: "-="
1043     STAR_EQ: "*="
1044     DIV_EQ: "/="
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2397
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2497
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2597
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
```

```

1026
1027     BOOLEAN: "true" | "false"
1028     NAME: /[-]?[a-zA-Z_][a-zA-Z0-9_]*/
1029     indexed_access: NAME "[" index_list "]"
1030     index_list: index_part ("," index_part)?
1031     index_part: expr?
1032     type: array_type | base_type type_suffix?
1033     BASE_TYPE: INT | REAL | VECTOR | MATRIX | SPARSE_MATRIX | ORDERED | SIMPLEX | BOOL
1034     base_type: BASE_TYPE
1035     type_suffix: "[" type_size ("," type_size)* "]"
1036     type_size: expr
1037     array_type: "array" "[" expr ("," expr)* "]"
1038     NUMBER: /-?[0-9]+(\.[0-9]+)?/
1039
1040     INT: "int"
1041     REAL: "real"
1042     VECTOR: "vector"
1043     MATRIX: "matrix"
1044     SPARSE_MATRIX: "sparse_matrix"
1045     ORDERED: "ordered"
1046     SIMPLEX: "simplex"
1047     BOOL: "bool"
1048
1049     %import common.ESCAPED_STRING
1050     %import common.WS
1051     %ignore WS
1052     MULTILINE_STRING_LITERAL: /"""(?:[^"\\"]|\\.|"""(?!"))*"""/
1053
1054     LPAR: "("
1055     RPAR: ")"
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079

```

B.3 EXAMPLE: LINEAR REGRESSION IN STATMODELDSL

The following example illustrates a simple linear regression task defined using our DSL:

Example: Simple Linear Regression in StatModelDSL

```

stat_model_spec linear_regression_example {
    meta {
        author: "StatBot";
        dsl_version: "1.0";
        model_version: "1.0";
        target_language: "stan";
    }

    documentation """
        Simple linear regression: predict y using x.
    """

    data {
        source N: int from csv(path="data.csv", head="num");
        source x: vector from csv(path="data.csv");
        source y: vector[N] from csv(path="data.csv");
    }

    transformed_data {
        x_centered: vector[N];
        x_mean: real;
        x_mean = mean(x);
        for (i in 1:N) {
            x_centered[i] = x[i] - x_mean;
        }
    }

    parameters {
        alpha: real ~ normal(0, 10);
        beta: real ~ normal(0, 5);
        sigma: real where sigma > 0 ~ exponential(1);
    }

    model {
        for (i in 1:N) {
            y[i] ~ normal(alpha + beta * x_centered[i], sigma);
        }
    }
}

```

```

1080
1081     inference {
1082         method: nuts;
1083         settings: {
1084             chains: 4;
1085             num_samples: 1000;
1086             num_warmup: 500;
1087         }
1088     }
1089
1090     output {
1091         monitor: [alpha, beta, sigma];
1092         summary_stats: [mean, q5, q95];
1093         diagnostics: [rhat];
1094         plots: [alpha, beta];
1095         export_results_to: "results_linear.csv";
1096     }
1097 }
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133

```

If we set “target_language” to “stan”, the transformed stan code is:

```

1095 Stan code for model inference
1096
1097 data {
1098     int N;
1099     vector[100] x;
1100     vector[N] y;
1101 }
1102 transformed data {
1103     vector[N] x_centered;
1104     real x_mean = mean(x);
1105     x_mean = mean(x);
1106     for (i in 1:N) {
1107         x_centered[i] = x[i] - x_mean;
1108     }
1109 }
1110 parameters {
1111     real alpha;
1112     real beta;
1113     real<lower=0.0> sigma;
1114 }
1115 model {
1116     alpha ~ normal(0, 10);
1117     beta ~ normal(0, 5);
1118     sigma ~ exponential(1);
1119     for (i in 1:N) {
1120         y[i] ~ normal(alpha + beta * x_centered[i], sigma);
1121     }
1122 }
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133

```

If we set “target_language” to “pymc”, the transformed PyMC code is:

```

1116
1117 PyMC code for model inference
1118
1119
1120 def build_and_sample(data, random_seed=None,
1121     return_inference_data=True, **sample_kwargs):
1122     import pymc as pm
1123     import numpy as np
1124     import aesara.tensor as at
1125     import arviz as az
1126     model = pm.Model()
1127     with model:
1128         N = pm.MutableData('N', data.get('N'))
1129         x = pm.MutableData('x', data.get('x'))
1130         y = pm.MutableData('y', data.get('y'))
1131         x_centered = x - at.mean(x)
1132         alpha = pm.Normal('alpha', 0, 10)
1133         beta = pm.Normal('beta', 0, 5)
1134         sigma = pm.Exponential('sigma', 1)
1135         mu = alpha + beta * x_centered
1136         pm.Normal('y', mu=mu, sigma=sigma, observed=y)
1137         chains = sample_kwargs.pop('chains', 4)
1138         draws = sample_kwargs.pop('draws', 1000)
1139         tune = sample_kwargs.pop('tune', 500)
1140         target_accept = sample_kwargs.pop('target_accept', 0.8)
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3079
3080
3081
3082
3083
3084
3085
3086
3087
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3148
3149
3150
3151
3152
3153
3154
3155
3156

```

```

1134
1135     trace = pm.sample(draws=draws, tune=tune, chains=chains,
1136     target_accept=target_accept, random_seed=random_seed, ** sample_kwargs)
1137     if return_inference_data:
1138         return az.from_pymc3(trace)
1139     return trace

```

C STATMODELCHATBOT

C.1 EXTRACTION SCHEMA

We design an extraction schema to facilitate the StatModelChatbot to verify all necessary task information. Here is the detailed schema we design:

```

1147
1148     {
1149         "data": {
1150             "variable_name": {
1151                 "variable_type": "",
1152                 "data_path": ""
1153             },
1154             ...
1155         },
1156         "transformed_data(optional)": {
1157             "variable_name": {
1158                 "variable_type": "",
1159                 "expression": ""
1160             },
1161             ...
1162         },
1163         "parameters": {
1164             "variable_name": {
1165                 "variable_type": "",
1166                 "distribution": "",
1167                 "constraint(optional)": ""
1168             },
1169             ...
1170         },
1171         "transformed_parameters(optional)": {
1172             "variable_name": {
1173                 "variable_type": "",
1174                 "expression": ""
1175             },
1176             ...
1177         },
1178         "model": [
1179             "expression1", "expression2", ...
1180         ],
1181         "inference": [
1182             "configuration1", "configuration2", ...
1183         ],
1184         "output": [
1185             "configuration1", "configuration2", ...
1186         ]
1187     }
1188

```

C.2 STATMODELCHATBOT WORKFLOW

As illustrated in Figure 2 (a), our StatModelChatbot goes through five sequential nodes to complete the entire schema and output a final task description. Here are the detailed prompts for our conversational agent.

Prompt for task definition

```

1183
1184     You are given a user's description of a dataset for a statistical modeling task. Please use
1185     a short paragraph to summarize what this task intends to do.
1186
1187     Output format:
1188     ---markdown
1189     <your answer>

```

```
1188
1189
1190
1191
1192     User:
1193     {user}
1194
1195     Example output:
1196     {example}
```

Prompt for data loading

```
1198
1199     You are an assistant who helps structure user descriptions into a predefined schema.
1200     The current task is about the **input data (data node)**.
1201
1202     The schema is defined as follows:
1203     {
1204         "data": {
1205             "variable_name": {
1206                 "variable_type": "",
1207                 "data_path": ""
1208             },
1209             ...
1210         },
1211         ...
1212     },
1213     prompt = prompt + f'''
1214
1215 Instructions:
1216 1. Carefully read the user's description:
1217     {user}
1218
1219 2. Fill in the schema above with as much information as possible.
1220     - `variable_name`: the name of the variable or dataset.
1221     - `variable_type`: the type of the variable (e.g., int, real, vector, array).
1222     - `data_path`: the exact path or identifier of the data
1223     (e.g., `benchmark/data/data1.csv`).
1224
1225 3. If the description provides enough information to complete a field, fill it in.
1226     If some required fields are missing, leave them as empty strings `""`.
1227
1228 4. Provide **two outputs**:
1229     - **Schema output** (inside ```json ... ```).
1230     - **Feedback in natural language** (inside ```markdown ```).
1231     - If all required information is present, the first line of Feedback must be `Enough`.
1232     - If some required information is missing, the first line must be `Not Enough`,
1233         followed by an explanation of what is missing and what the user should provide.
1234
1235 Make sure to always output both parts.
1236
1237 Examples:
1238 {example}
```

Prompt for variable specification

```
1227
1228
1229     You are an assistant that helps structure user descriptions into a predefined schema.
1230     The current task is about the **variable node**.
1231
1232     The schema is defined as follows:
1233     {
1234         "transformed_data(optional)": {
1235             "variable_name": {
1236                 "variable_type": "",
1237                 "expression": ""
1238             },
1239             ...
1240         },
1241         ...
1242         "parameters": {
1243             "variable_name": {
1244                 "variable_type": "",
1245                 "distribution": "", "
1246                 constraint(optional)": ""
1247             },
1248             ...
1249         }
1250     }
```

```

1242     },
1243     "transformed_parameters(optional)": {
1244         "variable_name": {
1245             "variable_type": "",
1246             "expression": ""
1247         },
1248         ...
1249     }
1250
1251     Instructions:
1252     1. Carefully read the user's description:
1253         {user}
1254
1255     2. Fill in the schema above with as much information as possible.
1256         - `parameters` is **required**. Each parameter should have:
1257             - `variable_name`: the name of the parameter.
1258             - `variable_type`: the type (e.g., int, real, vector, array).
1259             - `distribution`: the assumed prior distribution.
1260             - `constraint(optional)`: optional constraints (e.g., >0,
1261               between 0 and 1).
1262             - `transformed_data` and `transformed_parameters` are **optional**.
1263                 - If mentioned in the description, fill them in.
1264                 - If not mentioned, you may leave them out without asking the user.
1265
1266     3. If the description provides enough information to fully specify the required fields in
1267 `parameters`, fill them in.
1268         If some required fields are missing, leave them as empty strings `""`.
1269
1270     4. Provide **two outputs**: (the same as data loading)
1271
1272     Make sure to always output both parts.
1273     ---
1274     Examples:
1275         {example}
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295

```

Prompt for model definition

```

1270
1271
1272     You are an assistant that helps structure user descriptions into a predefined schema.
1273     The current task is about the **model node**.
1274
1275     The schema is defined as follows:
1276     {
1277         "model": [
1278             "expression1", "expression2", ...
1279         ]
1280     }
1281
1282     Instructions:
1283     1. Carefully read the user's description:
1284         {user}
1285
1286     2. Extract the part that describes the model (if any).
1287         - If the user mentions a model structure, equations, or likelihoods, summarize them as a
1288             list of expressions.
1289         - If the user does not provide any model information, leave the list empty.
1290
1291     3. Provide **two outputs**:
1292         - **Schema output** (inside ```json ... ```).
1293         - **Feedback in natural language** (inside ```markdown ... ```).
1294             - If model expressions are found, the first line of Feedback must be `Enough`, followed
1295                 by a short summary.
1296             - If no model information is found, the first line must be `Not Enough`, followed by a
1297                 clear request for the user to provide model details.
1298
1299     Make sure to always output both parts.
1300     ---
1301     Examples:
1302         {example}
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395

```

```

1296 Prompt for result configuration
1297
1298 You are an assistant that helps structure user descriptions into a predefined schema.
1299 The current task is about the inference and output nodes.
1300 The schema is defined as follows:
1301 {
1302     "inference": [
1303         "configuration1", "configuration2", ...
1304     ],
1305     "output": [
1306         "configuration1", "configuration2", ...
1307     ]
1308 }
1309
1310 Instructions:
1311 1. Carefully read the user's description:
1312     {user}
1313
1314 2. Identify whether the description includes any details about:
1315     - Inference (e.g., sampling method, number of iterations, variational inference,
1316     optimization settings, etc.).
1317     - Output (e.g., what results to return, summary statistics, predictions, posterior
1318     samples, etc.).
1319
1320 3. Provide two outputs:
1321     - Schema output (inside json ...), filling in the configurations if available
1322     - Feedback in natural language (inside markdown ...):
1323         - If inference and/or output details are found or the user says just default settings,
1324             the first line must be 'Enough', followed by a short summary.
1325         - If no inference/output information is found, the first line must be 'Not Enough',
1326             followed by a clear request for the user to specify inference and/or output
1327             configurations.
1328
1329 Make sure to always output both parts.
1330
1331 ---
1332 Examples:
1333 {example}

```

D AN ENTIRE EXAMPLE USING AUTOSTAT, FROM TASK SPECIFICATION TO DSL

First, the user can chat with the StatModelChatbot to specify the entire task, illustrated in Table 5. The StatModelChatbot will return the final output including the schema and the task description:

```

1332 Output schema
1333
1334 {
1335     "data": {
1336         "N": {
1337             "variable_type": "int",
1338             "data_path": "testdata.json"
1339         },
1340         "year": {
1341             "variable_type": "vector[int][N]",
1342             "data_path": "testdata.json"
1343         },
1344         "consumption": {
1345             "variable_type": "vector[N]",
1346             "data_path": "testdata.json"
1347         },
1348         "transformed_data": {
1349             "diff_consumption": {
1350                 "variable_type": "vector[N-1]",
1351                 "expression": "diff_consumption[i] = consumption[i+1] - consumption[i]"
1352                 for i in 1 to N-1"
1353             }
1354         },
1355         "parameters": {

```

1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364

Table 5: An example of the dialogue between user and StatModelChatbot.

1365	User	We're modeling year-to-year changes in electricity consumption using a simple Bayesian regression. After computing the first-order differences of consumption, we regress these differences on the corresponding years with a normal likelihood.
1366	Chatbot	Please provide the data paths for the electricity consumption differences and corresponding years data files.
1367	User	Our data is testdata.json, with an interger N and two vectors year and consumption.
1368	Chatbot	The description mentions using a normal likelihood for regression but doesn't specify the prior distributions for alpha, beta, and sigma. Please provide the prior distributions for these parameters (e.g., normal priors for alpha and beta, half-normal or half-cauchy for sigma).
1369	User	alpha is Normal(0, 10), beta Normal(0, 5), sigma is exp(1) with lower bound 0
1370	Chatbot	The description provides details about the Bayesian regression model structure (priors for alpha, beta, sigma, and likelihood) and the data source, but it doesn't specify any inference configurations (sampling method, number of iterations, chains, etc.) or output configurations (what results to return, summary statistics, predictions, or where to save outputs). Please specify your inference and output preferences, or indicate if you'd like to use default settings.
1371	User	For inference, I think just take default settings is ok. Use nuts sampling method, with 4 chains. I need 1000 warmup steps, then 1000 steps. For output, I need to see the median, ess, plots for alpha and beta. Remember to export results to this file: electricity_consumption_results.csv
1372		
1373		
1374		
1375		
1376		
1377		
1378		
1379		
1380		
1381		
1382		
1383		
1384		
1385		
1386		
1387		
1388		
1389		
1390		
1391		
1392		
1393		
1394		
1395		
1396		
1397		
1398		
1399		
1400		
1401		
1402		
1403		

```

1404
1405     "alpha": {
1406         "variable_type": "real",
1407         "distribution": "normal(0, 10)"
1408     },
1409     "beta": {
1410         "variable_type": "real",
1411         "distribution": "normal(0, 5)"
1412     },
1413     "sigma": {
1414         "variable_type": "real",
1415         "distribution": "exponential(1)",
1416         "constraint": "> 0"
1417     }
1418 },
1419 "model": ["diff_consumption[i] ~ normal(alpha + beta * year[i+1], sigma)
1420 for i in 1:(N-1)"]
1421 ],
1422 "inference": [
1423     "Use `nuts` method",
1424     "4 chains",
1425     "1000 samples after 1000 warmup iterations"
1426 ],
1427 "output": [
1428     "Report median as summary statistic",
1429     "Check effective sample size (ess) for diagnostics",
1430     "Generate plots for `alpha` and `beta`",
1431     "Export results to `electricity_consumption_results.csv`"
1432 ]
1433 }
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457

```

Output task definition

```

1429 You are provided with a dataset at `testdata.json`, which includes:
1430 - `N`: an integer representing the number of observations.
1431 - `year`: a vector of length `N` containing the observation years.
1432 - `consumption`: a vector of length `N` with consumption values (constrained to be greater
than 0).

1433 Additionally, you define a transformed variable:
1434 - `diff_consumption`: a vector of length `N-1`, calculated as
`consumption[i] - consumption[i-1]` for each `i` from 2 to N.

1435 You defined all required parameters clearly:
1436 - `alpha ~ normal(0, 10)`
1437 - `beta ~ normal(0, 5)`
1438 - `sigma > 0 ~ exponential(1)`

1439 You also introduced a transformed data variable:
1440 - `diff_consumption`: a vector of size `N-1`, where each element is
`consumption[i+1] - consumption[i]`.

1441 Model specification:
1442 - For each `i` in `1:(N-1)`, the differences in consumption are modeled as
`diff_consumption[i] ~ normal(alpha + beta * year[i+1], sigma)`.

1443 This means consumption changes are assumed to follow a normal distribution with mean
determined by a linear function of the year and standard deviation `sigma`.

1444 Inference configuration:
1445 - Use the `nuts` sampler with 4 chains.
1446 - Run 1000 warmup iterations followed by 1000 samples.

1447 Output configuration:
1448 - Report the **median** as the main summary statistic.
1449 - Check **effective sample size (ess)** for diagnostics.
1450 - Generate **plots for alpha and beta**.
1451 - Export results to **`electricity_consumption_results.csv`**.
1452
1453
1454
1455
1456
1457

```

Finally, our StatModelCopilot will generate the corresponding DSL code as follows:

Output task definition

```
1458
1459
1460 stat_model_spec electricity_consumption_model {
1461   meta {
1462     author: "StatBot";
1463     dsl_version: "1.0";
1464     model_version: "1.0";
1465     target_language: "stan";
1466   }
1467
1468   documentation """
1469   Bayesian regression model for year-to-year changes in electricity consumption.
1470   Models first-order differences of consumption regressed on years with normal likelihood.
1471   """
1472
1473   data {
1474     source N: int from json(path="testdata.json");
1475     source year: vector[N] from json(path="testdata.json");
1476     source consumption: vector[N] from json(path="testdata.json");
1477   }
1478
1479   transformed_data {
1480     consumption_diff: vector[N-1];
1481     year_diff: vector[N-1];
1482
1483     for (i in 1:(N-1)) {
1484       consumption_diff[i] = consumption[i+1] - consumption[i];
1485       year_diff[i] = year[i+1];
1486     }
1487   }
1488
1489   parameters {
1490     alpha: real ~ normal(0, 10);
1491     beta: real ~ normal(0, 5);
1492     sigma: real where sigma > 0 ~ exponential(1);
1493   }
1494
1495   model {
1496     for (i in 1:(N-1)) {
1497       consumption_diff[i] ~ normal(alpha + beta * year_diff[i], sigma);
1498     }
1499
1500   inference {
1501     method: nuts;
1502     settings: {
1503       chains: 4;
1504       num_samples: 2000;
1505       num_warmup: 1000;
1506     }
1507   }
1508
1509   output {
1510     monitor: [alpha, beta, sigma];
1511     summary_stats: [median, ess];
1512     diagnostics: [rhat];
1513     plots: [alpha, beta];
1514     export_results_to: "electricity_consumption_results.csv";
1515   }
1516 }
```

E STATMODEL DATASET

In this section, we will deliver more details about our dataset construction process, including prompt for data generation and prompt for data augmentation.

Prompt for data generation

You are a helpful DSL generator. I will provide you with:

- The grammar of the StatModelDSL,
- A few example DSL programs,
- A target usage scenario,
- And a set of configuration settings.

```

1512
1513
1514 Your task is to generate a complete StatModelDSL program that fits the given scenario and
1515 settings. Strictly follow the provided grammar and take inspiration from the examples.
1516 Additionally, create a detailed task description written from the user's perspective -
1517 imagine it's how someone would explain their modeling needs to an assistant.
1518
1519 You are allowed to define any `*.csv` or `*.json` data sources in your DSL program, but you
1520 do **not** need to generate their contents - we will provide the actual data files.
1521
1522 Please output your response in **exactly** the following format:
1523
1524 Description:
1525
1526     ```markdown
1527     <The user's prompt when he want copilot to generate this
1528     program. 1-3 sentences is ok.>
1529     ```
1530
1531 DSL code:
1532
1533     ```dsl
1534     <your complete DSL program here>
1535     ```
1536
1537
1538 **Inputs:***
1539
1540 DSL Grammar:
1541
1542     ```markdown
1543     {grammar}
1544     ```
1545
1546 DSL Program Examples:
1547
1548     ```dsl
1549     {examples}
1550     ```
1551
1552 Usage Scenario:
1553
1554 In the domain of **{domain}**, the task is **{task}**
```

Prompt for data augmentation

```

1543 You are a helpful assistant for StatModelDSL, a DSL similar to Stan. I will give you:
1544
1545 * A StatModelDSL code snippet (which will be compiled into Stan).
1546 * A natural-language task description written by a user.
1547 * A few task examples demonstrating how to rewrite the natural-language description to
1548 fully reflect the DSL specification.
1549
1550 Your job is to **rewrite the user's task description** so that it precisely
1551 corresponds to the provided DSL code. The rewritten description should specify:
1552
1553 1. All data inputs and their types (e.g., int, real, vector, array), and where they
1554 come from.
1555 2. Any derived (transformed) data or parameters and how they are computed.
1556 3. All model parameters and their prior distributions (with parameter values).
1557 4. The structure of the model (e.g., likelihoods, regression equations, etc.).
1558 5. Inference settings such as number of chains, number of samples, warm-up iterations.
1559
1560 #### Style instructions:
1561
1562 * The final description should **sound like a power user** of the tool giving precise
1563 instructions to the system, written in a natural, fluent tone, as if to a copilot.
1564 * Keep it concise but complete. Avoid vague language.
1565 * Don't explain what the DSL does - describe *what the user wants* in a way that fully
1566 specifies the model.
1567
1568 Please output your response using this format:
1569
1570 ---
1571
1572 Task description:
1573
1574     ```markdown
1575     <your rewritten description here>
```

```

1566
1567
1568
1569
1570     Input:
1571     Simple task description:
1572
1573     ```markdown
1574     {description}
1575     ```
1576
1577     DSL code:
1578
1579     ```dsl
1580     {dsl}
1581     ```

1582     Examples:
1583     {examples}
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619

```

F STATMODELCOPILOT

We choose Llama3.1-8B-Instruct (Dubey et al., 2024) as our base LLM to train our StatModelCopilot. The implementation details are shown in Table 6. All experiments are conducted on a single NVIDIA A40 GPU, supported by Llama-Factory (Zheng et al., 2024).

Table 6: Implementation details and training cost.

Stage	Learning Rate	Lora Rank	# Training Data	# Training Epochs	Training Time
One	1×10^{-4}	32	5064	5	3h
Two	5×10^{-5}	32	10907	3	7h

G MORE EXPERIMENTAL DETAILS

G.1 DETAILS ABOUT OUR TEST DATASET

Table 7: Details about our test set. All “lengths” mean the average lengths of all data. “Entire” means the entire test dataset.

	Simple	Medium	Complex	Entire
# Data	81	165	77	323
Instruction Length X_d	1676	1917	2153	1913
DSL code length Y	1091	1331	1615	1339

Table 7 demonstrates more details about our test dataset. We decompose our test set into 3 levels based on the length of the instructions to evaluate how different input lengths will affect the performance.

G.2 LLM-AS-A-JUDGE

For quantitative experiments, we leverage GPT-4o as a judge to list out the mismatching items between the description and the generated DSL/PPL code. Here is the prompt:

```

1620
1621     I will provide you with:
1622     - A DSL code program
1623     - The corresponding detailed description of the DSL code about the statistical modeling task
1624
1625     Please help me to check if the description exactly matches the code. You need to focus on
1626     each line of the description! Please list all the mismatches one by one.
1627
1628     If match, just answer: Match
1629     If not, answer in this format:
1630         Not match.
1631         1. In description, we need xxx, but in the code, xxx
1632         2. xxx
1633
1634         ---
1635
1636     DSL code:
1637         ```dsl
1638         {code}
1639         ```
1640
1641
1642     Description:
1643         ```markdown
1644         {description}
1645         ```

1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673

```

G.3 USER STUDY

Experimental settings. Each participant is asked to perform a simple code modification task in each of the different programming environments. The purpose is to give them a basic familiarity with the environment and ensure that they have carefully reviewed the provided code. After completing the tasks in all three environments, participants are asked to rank the code in terms of clarity and readability, and then indicate their preference—that is, which environment they would choose when encountering a similar task in the future. To ensure the validity of the results and avoid any order effects, the sequence in which participants worked with the three environments was randomized. The tasks they encountered were also assigned randomly, while maintaining similar levels of difficulty across conditions.

Participants. We collected a total of 17 responses. None of the participants had prior experience with PPLs such as Stan or PyMC, but all had a solid foundation in Python and data analysis, as well as a basic understanding of statistical concepts. From their responses, we can gain valuable insights into novice users’ preferences and perceptions of different programming environments.

Results Analysis. From the results shown in Figure 4, we can observe that: 1) Clarity: Over half of the participants favored our DSL, demonstrating that its design—by omitting verbose PPL code and general-purpose language code (here, Python)—is concise and highly readable. 2) Preference: Our DSL also achieved relatively high preference scores, indicating that for novices, this DSL format is indeed easier to use. However, compared with Stan, the advantage is not particularly large. We believe this is because participants were already very familiar with Python, and the statistical components in Stan were relatively minimal and straightforward. Additionally, the Stan code itself is fairly concise and easy to understand, which led many users to also find it accessible.

H REAL-WORLD EVALUATION

To further demonstrate the practicality and effectiveness of our StatModelDSL and the full AutoStat framework in real-world scenarios, we conduct a case study on two representative settings: 1) reproducing real examples from the Stan textbook, and 2) replicating experiments reported in published statistical research papers.

H.1 TASK COLLECTIONS

We collect real-world Bayesian modeling tasks from two types of scenarios:

1674 • **Textbook examples.** We collect real-world Stan-based code from the official Stan example repository³. These examples are drawn from three textbooks Gelman & Hill (2007); Lee & Wagenmakers (2014); Kéry & Schaub (2011). We first filter out overly simplistic or incomplete code, then randomly select 50 tasks from the remaining set. Each task is then converted into a complete natural language description using GPT-4o (Hurst et al., 2024), and subsequently implemented in StatModelDSL via our AutoStat framework.

1675 • **Research paper experiments.** We collect all papers published in **Bayesian Analysis**⁴ from 2022 onward. For each paper, we extracted all simulation experiments that either relied on Bayesian modeling or could be implemented using Bayesian approaches as a baseline. Using GPT-4o (Hurst et al., 2024), we converted these experiments into structured task descriptions, filtering out cases with overly vague descriptions or missing experimental details. From the remaining set, we randomly selected 50 tasks, which were implemented and simulated using StatModelDSL through our AutoStat framework. Because some experimental descriptions in the source papers are not sufficiently detailed and the resulting outputs span diverse dimensions, it is difficult to design rule-based checks to validate correctness. Therefore, our evaluation relies on a combination of syntactic validation, manual inspection, and LLM-as-a-judge assessment to determine whether the generated DSL faithfully matches the task requirements.

1676

1677

1678

1679

1680

1681

1682

1683

1684

1685

1686

1687

1688

1689

1690

1691 **All generated DSL programs are syntactically correct**, and the full set of programs is available
1692 in our official code repository.

1693

1694 **H.2 EXAMPLES**

1695 For textbook examples, here is a case:

1696

1697 **Task definition**

1698

1699 Develop a Bayesian hierarchical model using Stan to analyze data from multiple counties,
1700 focusing on estimating county-specific effects on a continuous outcome variable. The
1701 model includes hierarchical priors for county effects and uses NUTS sampling for inference,
1702 with key parameters including the overall mean, county-specific deviations, and their
1703 variances. The output should include plots of the primary parameters to assess their
1704 distributions and relationships.

1705

1706 (More details omitted)

1707

1708 **DSL implementation**

1709

```
1710 stat_model_spec hierarchical_county_model {
1711     meta {
1712         author: "Public Health Analyst";
1713         dsl_version: "1.0";
1714         model_version: "1.0";
1715         target_language: "stan";
1716     }
1717
1718     documentation """
1719     Bayesian hierarchical model for county-level data, using non-centered parameterization.
1720     """
1721
1722     data {
1723         source J: int from json(path="testdata.json");
1724         source N: int from json(path="testdata.json");
1725         source county: array[N] int from json(path="testdata.json");
1726         source y: vector[N] from json(path="testdata.json");
1727     }
1728
1729     parameters {
1730         eta: vector[J] ~ normal(0, sigma_eta);
1731         mu: real ~ normal(0, 1);
1732         sigma_eta: real where sigma_eta > 0 ~ uniform(0, 100);
1733         sigma_y: real where sigma_y > 0 ~ uniform(0, 100);
1734     }
1735 }
```

1736

1737 ³<https://github.com/stan-dev/example-models>

1738 ⁴<https://projecteuclid.org/journals/bayesian-analysis>

```

1728
1729
1730     transformed_parameters {
1731         eta_adj: vector[J];
1732         mean_eta: real;
1733         mu_adj: real;
1734         for (j in 1:J) {
1735             eta_adj[j] = eta[j] - mean(eta);
1736             mean_eta = mean(eta);
1737         }
1738         mu_adj = 100 * mu + mean_eta;
1739         for (i in 1:N) {
1740             y_hat: vector[N];
1741             for (j in 1:J) {
1742                 y_hat[i] = 100 * mu + eta[county[i]];
1743             }
1744         }
1745     }
1746
1747     model {
1748         mu ~ normal(0, 1);
1749         sigma_eta ~ uniform(0, 100);
1750         sigma_y ~ uniform(0, 100);
1751         for (j in 1:J) {
1752             eta[j] ~ normal(0, sigma_eta);
1753         }
1754         for (i in 1:N) {
1755             y[i] ~ normal(y_hat[i], sigma_y);
1756         }
1757     }
1758
1759     inference {
1760         method: nuts;
1761         settings: {
1762             chains: 4;
1763             num_samples: 1000;
1764             num_warmup: 500;
1765         }
1766     }
1767
1768     output {
1769         monitor: [mu, sigma_eta, sigma_y, eta];
1770         summary_stats: [mean, q5, q95];
1771         diagnostics: [rhat];
1772         plots: [mu, sigma_eta, sigma_y];
1773         export_results_to: "results_hierarchical_county.csv";
1774     }
1775
1776 }
```

1762 This dsl implementation perfectly aligns with the task description and can be converted to the
1763 ground-truth stan code.

1764 For research paper examples, here is a case: In Section S4 of the appendix in Beraha et al. (2024),
1765 the authors conducted a Bayesian analysis on a mixture of Gaussians to compare the performance of
1766 different sampling methods. Here, we replicate this experiment using StatModelDSL, performing
1767 MCMC sampling with Stan. The corresponding DSL code is as follows:

```
1768
1769     stat_model_spec mixture_gaussian {
1770         meta {
1771             author: "User";
1772             dsl_version: "1.0";
1773             model_version: "1.0";
1774             target_language: "stan";
1775         }
1776
1777         documentation """
1778             We want to build a mixture-of-Gaussian distribution and test
1779             the inference success rate and time cost of MCMC sampling.
1780         """
1781
1782         data {
1783             source N: int where N > 1 from json(path="testdata.json");
1784             source K: int where K > 1 from json(path="testdata.json");
1785             source y: vector[N] from json(path="testdata.json");
1786         }
1787     }
```

```

1782
1783
1784     parameters {
1785         theta: simplex[K] ~ dirichlet(rep_vector(1.0, K));
1786         sigma: vector[K] ~ cauchy(0, 2.5);
1787         mu: vector[K] ~ normal(0, 5);
1788     }
1789
1790     model {
1791         // Likelihood: mixture density
1792         for (n in 1:N) {
1793             lps: vector[K];
1794             for (k in 1:K) {
1795                 lps[k] = log(theta[k]) +
1796                     normal_lpdf(y[n] | mu[k], sigma[k]);
1797             }
1798             target += log_sum_exp(lps); // mixture log-likelihood
1799         }
1800     }
1801
1802     inference {
1803         method: nuts;
1804         settings: {
1805             chains: 4;
1806             num_samples: 2000;
1807             num_warmup: 1000;
1808         }
1809     }
1810
1811     output {
1812         monitor: [theta];
1813         summary_stats: [mean, q5, q95];
1814         diagnostics: [rhat, ess];
1815         export_results_to: "results.csv";
1816         posterior_predictive_checks: true;
1817         age_group_differences: true;
1818         credible_intervals: [0.9, 0.95];
1819     }
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835

```

Our experimental results show that when $n = 100$, the effective sample size (ESS) is 22.33 with a sampling time of 2.58s; when $n = 250$, the ESS decreases to 14.61 with a sampling time of 3.78s. Compared with Figure 1 in Section S4 (Beraha et al., 2024), these results fall within a consistent and realistic range.