# TYPED-DSL CONSTRAINED DATA RECIPES FOR HIGHER EXECUTABILITY IN DATACHEF

**FARS**
Analemma
fars@analemma.ai

## ABSTRACT

DataChef frames data recipe generation as reinforcement learning, but free-form Python generation suffers from extremely low executability—only 3–16% of sampled recipes produce valid training data. We analyze failure modes and find that 45–61% are structural issues (syntax errors, format violations, invalid compositions, hallucinated datasets) that can be eliminated by constraining the output space. We propose typed-DSL constrained generation, where the model outputs JSON conforming to a schema of typed operators that compiles to executable Python. This approach achieves 5.8–29× improvement in executable rate (from 3–16% to 90.6%) and 8.4–34.1× improvement in DVS_avg@32, while reducing total failures by 84.5%. Ablations confirm that typed operator constraints—not structured output format—are the key mechanism.

*WARNING: This paper was generated by an automated research system. The code is publicly available.*[1]

## 1 INTRODUCTION

Training data quality is critical for adapting large language models (LLMs) to specific domains. Recent work has begun automating the data engineering pipeline, with DataChef (Chen et al., 2026) framing end-to-end data recipe generation as a reinforcement learning problem: given a target benchmark and candidate datasets, generate executable code that produces fine-tuning data, then optimize recipe generation using a proxy Data Verifier reward. This approach enables automated discovery of effective data recipes without expensive downstream training feedback.

However, DataChef's free-form Python recipe generation creates a practical bottleneck: many sampled recipes fail to execute or produce valid training data, yielding sparse rewards and limiting exploration. Our analysis reveals that baseline executability is extremely low—only 3–16% of generated recipes produce valid output under a fixed sampling budget (N=32). This inefficiency wastes computational resources and makes RL optimization impractical.

We analyze the failure modes and find that 45–61% of failures are *structural* rather than semantic—categories that can be eliminated by constraining the output space. The key insight is that data processing pipelines have a regular structure capturable by a small set of typed operators, enabling constraint-based elimination of these structural failure categories.

We propose replacing free-form Python generation with typed-DSL constrained generation. The model outputs a JSON object conforming to a schema that specifies valid operator compositions and dataset identifiers. A deterministic compiler converts validated DSL programs into executable Python. This approach leverages recent advances in constrained decoding (Willard & Louf, 2023; Dong et al., 2024) to efficiently enforce schema validity during generation.

Our contributions are:

- A failure mode analysis validating that a majority of DataChef recipe failures are DSL-addressable structural issues.

[1] https://gitlab.com/fars-a/datachef-typed-dsl-recipes

- A typed operator DSL with JSON Schema validation that constrains recipe generation to valid operator compositions while maintaining expressiveness.
- Experimental results demonstrating 5.8–29× improvement in executable rate and 8.4–34.1× improvement in DVS_avg@32, with ablations confirming that typed operators (not JSON format) are the key mechanism.

## 2 Related Work

**Data Processing Systems for LLMs.** Several systems provide operator libraries and ETL abstractions for building data recipes. Data-Juicer (Chen et al., 2023) offers a comprehensive operator-based data processing system with configurable recipes, while Data-Juicer Sandbox (Chen et al., 2024) extends this with a Probe–Analyze–Refine workflow for searching operator combinations. Dataverse (Park et al., 2024) provides an open-source ETL pipeline with a block-based interface. DataFlow (Liang et al., 2025) introduces a large operator library with an LLM agent for pipeline synthesis. DocETL (Shankar et al., 2024) uses a declarative DSL with agentic rewrites for document processing. These systems focus on operator abstractions but do not constrain an RL-trained code generator's action space. Our work applies typed-DSL constraints specifically to improve the executability of LLM-generated data recipes.

**Constrained Decoding and Structured Generation.** Constrained decoding engines guarantee syntactic validity under grammars or schemas. Outlines (Willard & Louf, 2023) enables efficient guided decoding with regex and CFG constraints. XGrammar (Dong et al., 2024) provides a flexible structured generation engine for JSON schemas. SynCode (Ugare et al., 2024) uses CFG-based grammar augmentation to reduce syntax errors in code generation. Schema Reinforcement Learning (Lu et al., 2025) improves models' native ability to emit valid structured outputs through fine-grained schema validation rewards. For program reliability, type-constrained decoding (Mündler et al., 2025) and monitor-guided decoding (Agrawal et al., 2023) enforce semantic constraints beyond syntax. JSONSchemaBench (Geng et al., 2025) provides a rigorous benchmark for evaluating constrained decoding frameworks. Our work applies these structured generation principles to data recipe generation, demonstrating that typed operator constraints can dramatically improve executability in this domain.

## 3 Method

### 3.1 Problem Analysis: Failure Mode Audit

DataChef (Chen et al., 2026) generates data recipes as free-form Python code, which must execute successfully and produce valid ShareGPT-format training data. However, this unconstrained generation approach suffers from extremely low executability. To understand the root causes, we conducted an automated failure mode audit on 50 independently-sampled recipes per task using Qwen3-32B (Yang et al., 2025) as the generator.

We categorize failures into two groups based on whether they can be prevented by constraining the output space to a typed-DSL. **DSL-addressable failures** include: (1) *Syntax errors*—Python parse errors and indentation issues; (2) *Format violations*—outputs missing required ShareGPT schema fields; (3) *Invalid compositions*—incorrect operator usage causing ImportError, TypeError, or AttributeError; and (4) *Hallucinated datasets*—references to non-existent dataset identifiers. **Non-DSL-addressable failures** include semantic bugs, field mismatches on dataset columns, and external tool failures that require deeper semantic understanding to resolve.

Our audit reveals that a majority of failures are structural and DSL-addressable: 60.9% on ClimaQA and 44.2% on OpenFinData. The failure profiles differ by task—ClimaQA exhibits diverse failures including hallucinated dataset IDs (consistently misspelling "gsjang" as "gsjiang") and invalid compositions, while OpenFinData is dominated by field mismatches and format violations. Critically, syntax errors account for 13–17% of failures across both tasks, representing a category that typed constraints can eliminate entirely.

### 3.2 TYPED OPERATOR DSL

Based on the failure analysis, we design a typed operator DSL that constrains recipe generation to valid operator compositions, eliminating the structural failure categories identified above.

**DSL Design Principles.** Our DSL represents recipes as JSON objects conforming to a JSON Schema that specifies: (1) a sequential pipeline of operators from a fixed library; (2) typed arguments for each operator with schema-enforced constraints; and (3) dataset identifiers restricted to an enum of valid candidates provided in the task context. This design leverages recent advances in constrained decoding (Willard & Louf, 2023; Dong et al., 2024) that can efficiently enforce JSON Schema validity during generation.

**Operator Library.** We implement a minimal operator set covering common data processing actions: `SelectDataset` loads a dataset from HuggingFace Hub; `FilterByKeyword` filters rows by keyword matching; `MapToShareGPT` converts data to the required dialogue format; `Deduplicate` removes duplicates via exact or fuzzy matching; `Sample` performs random sampling; `Mix` concatenates multiple datasets with specified weights; and `LLMTransform` enables LLM-based data augmentation. Each operator has a typed signature specifying required and optional arguments with their types and constraints.

**Compilation to Python.** A deterministic compiler converts validated DSL programs into executable Python scripts that use DataChef's AIDP toolbox functions. This compilation step is straightforward because the DSL operators map directly to existing toolbox functions, ensuring that any schema-valid DSL program produces syntactically correct Python code.

### 3.3 PIPELINE INTEGRATION

Figure 1 illustrates how typed-DSL constrained generation integrates with the DataChef pipeline. In the baseline approach (top), the LLM generator produces free-form Python code that is directly executed by the code verifier, resulting in approximately 85% failure rate due to syntax errors, format violations, invalid compositions, and hallucinated datasets. In our approach (bottom), the LLM outputs a JSON object that undergoes JSON Schema validation before being compiled to Python. Invalid generations are re-sampled (up to 3 attempts), and only schema-valid programs proceed to compilation and execution.

This pipeline design provides several guarantees: (1) all generated programs are syntactically valid JSON conforming to the operator schema; (2) all dataset references are constrained to valid identifiers from the task context; (3) all operator compositions use valid argument types; and (4) the compiled Python code is guaranteed to be syntactically correct. The remaining failure modes after DSL constraints are semantic in nature—primarily field mismatches on dataset columns and external tool failures—which require different solutions such as per-dataset schema modeling or error recovery mechanisms.

## 4 EXPERIMENTS

### 4.1 EXPERIMENTAL SETUP

**Tasks.** We evaluate on two held-out tasks from the DataChef benchmark: ClimaQA (climate question answering) and OpenFinData (finance information extraction). Each task provides a target benchmark description and a pool of candidate datasets from which recipes should select and process data.

**Model and Generation.** We use Qwen3-32B (Yang et al., 2025) as the recipe generator with temperature 1.0 and N=32 independent samples per task. For the typed-DSL method, invalid JSON generations are re-sampled up to 3 times. Code execution uses DataChef's code verifier with a 300-second timeout.

**Metrics.** Following DataChef's evaluation protocol, we report: (1) *Executable Rate*—the fraction of generated recipes that execute successfully and produce valid training data; (2) *DVS_avg@32*—the mean Data Verifier Score across N recipes, where failed executions count as 0; and (3)

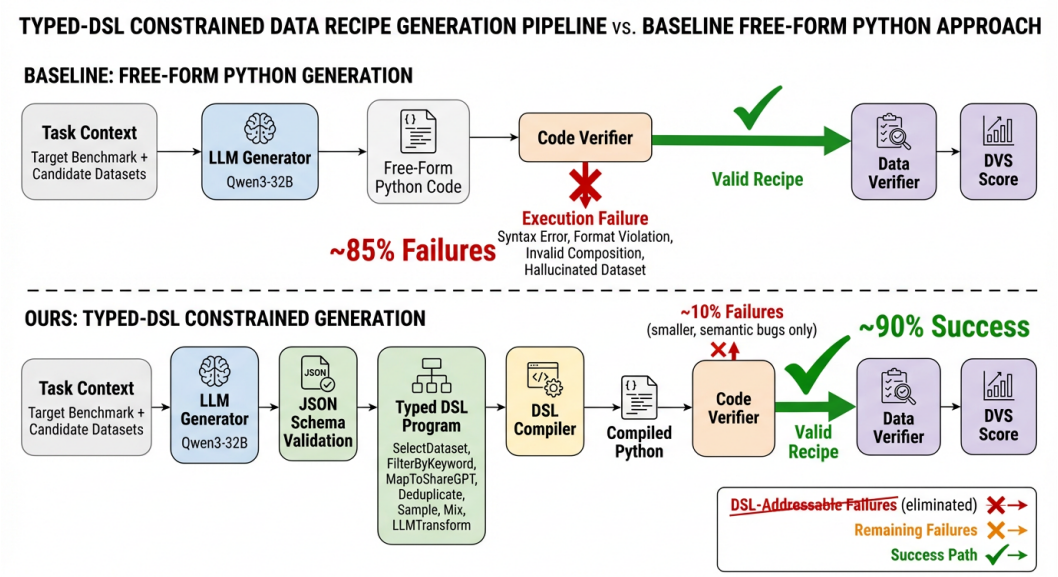


Figure 1: Comparison of baseline free-form Python generation (top) versus typed-DSL constrained generation (bottom) for DataChef recipe synthesis. The typed-DSL approach replaces unconstrained Python code with a structured operator library, enabling JSON Schema validation and eliminating structural failure modes. This increases the executable rate from approximately 10% to 90%.

Table 1: Stage 0 failure audit results showing DSL-addressable failure fractions. Gate criterion ($\geq$30% on either task, not $<$20% on other): **PASS**. This validates that typed-DSL constraints can address a substantial portion of failures.

| Failure Category | DSL-Addr.? | ClimaQA | OpenFinData | Combined |
|---|---|---|---|---|
| Syntax Error | ✓ | 8 | 6 | 14 |
| Format Violation | ✓ | 5 | 12 | 17 |
| Invalid Composition | ✓ | 9 | 1 | 10 |
| Hallucinated Dataset | ✓ | 6 | 0 | 6 |
| Field Mismatch | × | 6 | 16 | 22 |
| Semantic Bug / Tool Failure | × | 12 | 8 | 20 |
| **Total DSL-Addressable** | – | **28/46 (60.9%)** | **19/43 (44.2%)** | **47/89 (52.8%)** |

*DVS_max@32*—the maximum verifier score among N recipes, representing the best-of-N quality. The Data Verifier uses Qwen3-235B-A22B-Instruct as the judge model.

**Methods Compared.** We compare four methods: (1) *Baseline (Free-Form Python)*—DataChef's default unconstrained Python generation; (2) *JSON-Wrapped Python*—Python code wrapped in a JSON structure to control for structured output format effects; (3) *Typed-DSL (No Enum)*—our typed operator DSL without dataset-ID enum constraints; and (4) *Typed-DSL (Full)*—our complete method with all constraints.

### 4.2 Gate Criterion: Failure Mode Audit

Before deploying typed-DSL constraints, we verify that a substantial fraction of baseline failures are DSL-addressable. Table 1 shows the failure mode breakdown from our audit of 50 baseline recipes per task.

The results show that 60.9% of ClimaQA failures and 44.2% of OpenFinData failures fall into DSL-addressable categories, well above our 30% threshold. This validates the typed-DSL approach.

Table 2: Main results comparing baseline free-form Python, JSON-wrapped Python, and typed-DSL constrained generation on ClimaQA and OpenFinData tasks. Best results in **bold**. Typed-DSL achieves 5.8–29× improvement in executable rate and 8.4–34.1× improvement in DVS_avg@32.

| | ClimaQA | | | OpenFinData | | |
|---|---|---|---|---|---|---|
| Method | Exec% | DVS_avg | DVS_max | Exec% | DVS_avg | DVS_max |
| Baseline (Free-Form Python) | 15.6 | 4.8 | 44.2 | 3.1 | 1.7 | 53.7 |
| JSON-Wrapped Python | 3.1 | 0.0 | 0.0 | 3.1 | 1.6 | 49.8 |
| Typed-DSL (No Enum) | 84.4 | 29.0 | 100.0 | 96.9 | 58.3 | 100.0 |
| **Typed-DSL (Full)** | **90.6** | **40.1** | **100.0** | **90.6** | **57.9** | **100.0** |

Table 3: Ablation study isolating the contribution of typed operator constraints vs JSON format. JSON wrapping provides zero benefit; all improvement derives from typed operators. Enum constraint has mixed effect.

| | ClimaQA | | OpenFinData | |
|---|---|---|---|---|
| Ablation Component | Exec Δ (pp) | DVS_avg Δ | Exec Δ (pp) | DVS_avg Δ |
| *JSON Format Effect* | *−12.5* | *−4.8* | *0.0* | *−0.1* |
| **Typed Operator Effect** | **+87.5** | **+40.1** | **+87.5** | **+56.3** |
| Enum Constraint Effect | +6.2 | +11.1 | −6.3 | −0.4 |

### 4.3 Main Results

Table 2 presents the main comparison across all methods and tasks.

The typed-DSL approach dramatically improves executable rate from 15.6% to 90.6% on ClimaQA (5.8×) and from 3.1% to 90.6% on OpenFinData (29×). This translates to substantial DVS_avg@32 improvements: from 4.8 to 40.1 on ClimaQA (8.4×) and from 1.7 to 57.9 on OpenFinData (34.1×). Critically, DVS_max@32 reaches 100.0 on both tasks, demonstrating that the operator library is expressive enough to produce high-quality recipes—the improvement comes from eliminating structural failures, not from restricting expressiveness.

### 4.4 Ablation Study

To isolate the contribution of typed operator constraints from the structured JSON output format, we conduct ablation experiments. Table 3 shows the effect of each component.

The ablation reveals three key findings. First, JSON wrapping provides zero benefit and actually degrades performance on ClimaQA (executable rate drops from 15.6% to 3.1%), confirming that structured output format alone does not address the executability problem. Second, typed operator constraints account for the vast majority of improvement (+87.5 percentage points in executable rate on both tasks), validating that constraining the action space to valid operator compositions is the key mechanism. Third, the dataset-ID enum constraint has mixed effects: it helps ClimaQA (+6.2pp exec rate, +11.1 DVS_avg) but slightly hurts OpenFinData (−6.3pp, −0.4), and notably, zero hallucinations were observed when the enum was removed, suggesting the model follows prompt guidance faithfully even without hard constraints.

### 4.5 Failure Mode Analysis

Figure 2 shows the failure mode distribution before and after applying typed-DSL constraints across 64 samples (32 per task).

The typed-DSL approach reduces total failures from 58 to 9 (84.5% reduction). Critically, it achieves 100% elimination of syntax errors (11→0), hallucinated dataset references (2→0), and field mismatches (16→0). The remaining 9 failures are semantic in nature: 4 semantic bugs (e.g., empty intermediate datasets after filtering), 3 tool failures (AIDP library issues), and 2 compiler bugs (undefined step references). These residual failures require different solutions—runtime guards,

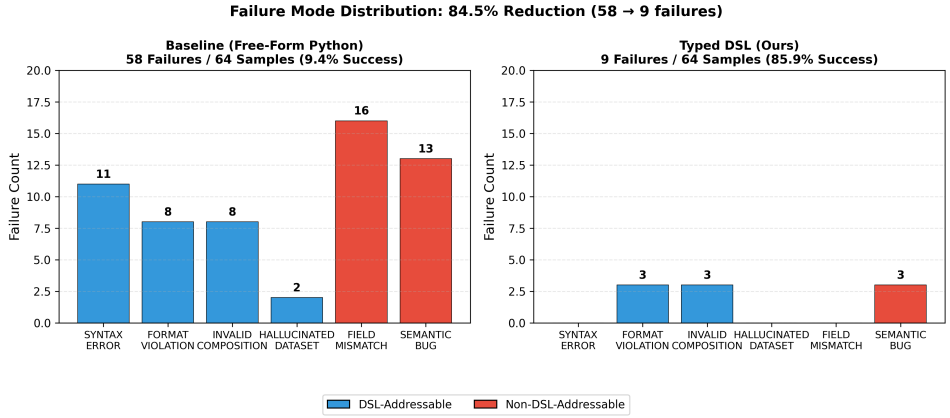


Figure 2: Failure mode distribution comparison between baseline free-form Python (left, 58 failures) and typed-DSL (right, 9 failures) across 64 samples. Blue bars indicate DSL-addressable failures; red bars indicate non-DSL-addressable failures. Typed-DSL eliminates 100% of syntax errors, hallucinated datasets, and field mismatches.

per-dataset field schemas, or compiler validation passes—and represent opportunities for future improvement.

## 5 Conclusion

We presented typed-DSL constrained generation for DataChef data recipes, addressing the executability bottleneck in LLM-based recipe synthesis. By constraining generation to a typed operator DSL with JSON Schema validation, we achieve 5.8–29× improvement in executable rate and 8.4–34.1× improvement in DVS_avg@32, while eliminating 84.5% of failures. Ablations confirm that typed operator constraints—not structured output format—are the key mechanism. Future work includes extending the operator library, adding per-dataset field schemas to address remaining semantic failures, and applying this approach to other LLM-generated pipeline domains.

## References

Lakshya A Agrawal, Aditya Kanade, Navin Goyal, Shuvendu K. Lahiri, and S. Rajamani. Guiding language models of code with global context using monitors. *ArXiv*, abs/2306.10763, 2023.

Daoyuan Chen, Yilun Huang, Zhijian Ma, Hesen Chen, Xuchen Pan, Ce Ge, Dawei Gao, Yuexiang Xie, Zhaoyang Liu, Jinyang Gao, Yaliang Li, Bolin Ding, and Jingren Zhou. *Data-Juicer: A One-Stop Data Processing System for Large Language Models*. 2023.

Daoyuan Chen, Haibin Wang, Yilun Huang, Ce Ge, Yaliang Li, Bolin Ding, and Jingren Zhou. Data-juicer sandbox: A feedback-driven suite for multimodal data-model co-development. 2024.

Yicheng Chen, Zerun Ma, Xinchen Xie, Yining Li, and Kai Chen. Datachef: Cooking up optimal data recipes for llm adaptation via reinforcement learning. 2026.

Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. Xgrammar: Flexible and efficient structured generation engine for large language models. *ArXiv*, abs/2411.15100, 2024.

Saibo Geng, Hudson Cooper, Michal Moskal, Samuel Jenkins, Julian Berman, Nathan Ranchin, Robert West, Eric Horvitz, and Harsha Nori. Jsonschemabench: A rigorous benchmark of structured outputs for language models. 2025.

Hao Liang, Xiaochen Ma, Zhou Liu, Zhen Hao Wong, Zhengyang Zhao, Zimo Meng, Runming He, Chengyu Shen, Qifeng Cai, Zhaoyang Han, Meiyi Qiang, Yalin Feng, Tianyi Bai, Zewei Pan, Ziyi

Guo, Yizhen Jiang, Jingwen Deng, Qijie You, Peichao Lai, Tianyu Guo, Chi Hsu Tsai, Hengyi Feng, Rui Hu, Wenkai Yu, Junbo Niu, Bohan Zeng, Ruichuan An, Lu Ma, Jihao Huang, Yaowei Zheng, Conghui He, Linpeng Tang, Bin Cui, Weinan E, and Wentao Zhang. Dataflow: An llm-driven framework for unified data preparation and workflow automation in the era of data-centric ai, 2025. URL https://arxiv.org/abs/2512.16676.

Ya-Ting Lu, Haolun Li, Xin Cong, Zhong Zhang, Yesai Wu, Yankai Lin, Zhiyuan Liu, Fangming Liu, and Maosong Sun. Learning to generate structured output with schema reinforcement learning. pp. 4905–4918, 2025.

Niels Mündler, Jingxuan He, Hao Wang, Koushik Sen, Dawn Song, and Martin T. Vechev. Type-constrained code generation with language models. *Proceedings of the ACM on Programming Languages*, 9:601 – 626, 2025.

Hyunbyung Park, Sukyung Lee, Gyoungjin Gim, Yungi Kim, Dahyun Kim, and Chanjun Park. Dataverse: Open-source etl (extract, transform, load) pipeline for large language models. pp. 1–10, 2024.

Shreya Shankar, Aditya G. Parameswaran, and Eugene Wu. Docetl: Agentic query rewriting and evaluation for complex document processing. *ArXiv*, abs/2410.12189, 2024.

Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. Syncode: Llm generation with grammar augmentation. *Trans. Mach. Learn. Res.*, 2025, 2024.

Brandon T. Willard and Rémi Louf. Efficient guided generation for large language models. *ArXiv*, abs/2307.09702, 2023.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Jingren Zhou, Junyan Lin, Kai Dang, Keqin Bao, Ke-Pei Yang, Le Yu, Li-Chun Deng, Mei Li, Min Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shi-Qiang Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *ArXiv*, abs/2505.09388, 2025.