

TEFD: A Benchmark for Natural Language to Flux Query Generation in Time-Series Databases

Xuefeng Wu
College of Computer Science and
Software Engineering, Shenzhen
University
Shenzhen, China
2410105039@mails.szu.edu.cn

Yuanfeng Song
Independent Researcher
Shanghai, China
songyf@outlook.com

Jiawei Wen
College of Computer Science and
Software Engineering, Shenzhen
University
Shenzhen, China
wenjiawei2023@email.szu.edu.cn

Dandan Huang
College of Computer Science and
Software Engineering, Shenzhen
University
Shenzhen, China
2400101071@mails.szu.edu.cn

Raymond Chi-Wing Wong
Department of Computer Science and
Engineering (CSE), The Hong Kong
University of Science and Technology
Hong Kong, China
raywong@cse.ust.hk

Haodi Zhang
College of Computer Science and
Software Engineering, Shenzhen
University
Shenzhen, China
zhanghd.ustc@gmail.com

Abstract

The proliferation of IoT and real-time monitoring has established Time-Series Databases (TSDBs) like InfluxDB as critical infrastructure. However, their functional query languages (e.g., Flux) present a steep learning curve, hindering data accessibility for non-experts. While Natural Language Interfaces (NLIs) offer a potential solution, the domain of **Text-to-Flux** is stalled by a critical bottleneck: the complete absence of diverse, high-quality paired benchmarks. To address this, we introduce the Text-to-Flux task and propose **FluxEngine**, a novel automated **pipeline** for dataset construction. Unlike static generation methods used in Text-to-SQL, our framework features a *Self-Sustaining Live Data Context* that utilizes background tasks to perpetually generate fresh data, ensuring that queries involving relative time windows (e.g., “past hour”) remain executable and valid indefinitely. Using this framework, we construct and release **TEFD** (TExt-to-Flux Dataset), the first large-scale benchmark for this task. We further define execution-based evaluation metrics tailored for time-series validity. This work provides the essential infrastructure to foster future research in NLI for TSDBs. To facilitate reproducibility and future research, our dataset and benchmark code are publicly available at <https://github.com/gta886/TEFD-Benchmark>.

CCS Concepts

• **Information systems** → **Query languages**; *Data mining*; • **Computing methodologies** → **Natural language processing**.

Keywords

Time-Series Databases, Text-to-Flux, Natural Language Interfaces, Dataset Generation, InfluxDB

ACM Reference Format:

Xuefeng Wu, Yuanfeng Song, Jiawei Wen, Dandan Huang, Raymond Chi-Wing Wong, and Haodi Zhang. 2026. TEFD: A Benchmark for Natural Language to Flux Query Generation in Time-Series Databases. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD 2026)*, August 9–13, 2026, Jeju Island, Republic of Korea. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3770855.3817508>

1 Introduction

The explosive growth of IoT systems and real-time monitoring has cemented Time-Series Databases (TSDBs) like InfluxDB [10] as critical infrastructure. As highlighted in recent surveys on AIOps in the LLM era [29], the ability to autonomously analyze telemetry data is essential for next-generation incident response and system observability. To handle high-velocity data, these systems employ specialized query languages such as **Flux**. Unlike the ubiquitous declarative syntax of SQL, Flux is a functional data scripting language that processes streams through a pipeline of transformations using the pipe-forward operator (i.e., `|>`). While this paradigm offers granular control over data streams, it creates a significant *cognitive barrier* for users unfamiliar with functional dataflow paradigms [7, 17]. Domain experts accustomed to SQL often struggle to adapt to Flux’s functional logic, creating a bottleneck in data accessibility for complex root-cause analysis.

Natural Language Interfaces (NLIs) driven by Large Language Models (LLMs) offer a promising solution to bridge this gap. However, the efficacy of LLMs is fundamentally constrained by data availability. The landscape of Text-to-Query benchmarking has recently undergone a paradigm shift, moving from academic relational snapshots (e.g., Spider 1.0 [27]) to complex, enterprise-level workflows like Spider 2.0 [12] and execution-heavy BI scenarios like BIRD [14]. Simultaneously, specialized NLIs have emerged for monitoring DSLs such as PromQL [28], LogQL [20], and Kusto (KQL) [23], while the Text-to-NoSQL domain has matured with benchmarks like TEND [16] and MultiTend [19].

We introduce the **Text-to-Flux** problem, which involves translating natural language into Flux queries. While general text-to-query research has matured significantly, the Flux domain remains



This work is licensed under a Creative Commons Attribution 4.0 International License. *KDD 2026, Jeju Island, Republic of Korea.*

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2259-2/2026/08

<https://doi.org/10.1145/3770855.3817508>

largely unexplored. This gap is particularly critical because, as empirical studies suggest, general-purpose LLMs often struggle to understand the nuanced temporal logic and anomalies inherent in time-series data [3, 6, 30]. Without high-quality paired data, even state-of-the-art models frequently hallucinate syntax or fail to synthesize the deep, state-dependent functional chains required for Flux pipelines. Constructing a benchmark for Flux presents unique challenges that render static SQL methodologies inadequate: i) *Temporal Obsolescence*: Time-series queries are inherently dynamic (e.g., `range(start: -1h)`). In a static dataset snapshot, such queries become “stale” and invalid as time progresses, returning empty results that make execution-based evaluation impossible. ii) *Functional Pipeline Complexity*: Unlike the nested relational algebra of SQL, Flux requires constructing deep functional chains (e.g., `filter` $\rightarrow$ `window` $\rightarrow$ `mean` $\rightarrow$ `map`). Validating these requires checking the data flow state, not just final output equivalence.

These challenges highlight a structural mismatch between static evaluation paradigms and the inherently dynamic nature of time-series systems. Most existing Text-to-Query benchmarks are evaluated on fixed database snapshots and implicitly assume that query semantics are time-invariant. In streaming environments, however, queries are intrinsically time-relative and state-dependent, causing this assumption to break. As a result, execution-based evaluation becomes unstable: semantically correct programs may return empty or misleading results due solely to temporal drift rather than modeling errors. This observation indicates that a practical Text-to-Flux benchmark must satisfy two core requirements. Relative-time queries must remain executable and semantically meaningful as time advances, and evaluation outcomes must be deterministic and reproducible across runs and models. Static corpora and snapshot-based benchmarks fail to meet these requirements simultaneously.

To address this gap, we propose an evaluation paradigm that couples a continuously refreshed data context with deterministic temporal normalization. This design preserves the semantics of time-relative queries while enabling reliable, reproducible execution-based evaluation over long time horizons.

In short, our key contributions are as follows:

- **The FluxEngine Framework**: We propose a novel, reproducible orchestration engine for dataset construction. A key innovation is our **Self-Sustaining Data Context**, which utilizes background tasks to continuously inject fresh data. This ensures that relative time queries remain executable and semantically valid indefinitely.
- **The TEFD Dataset**: We release **TEFD (TExt-to-Flux Dataset)**, the first large-scale, verified benchmark for this domain. It covers diverse patterns, from basic filtering to complex windowing, joins, and map transformations.
- **Task Definition & Metrics**: We formally define the Text-to-Flux task and propose **Time-Series Execution Accuracy (TS-EX)** as the gold standard for evaluation, moving beyond simple string matching to verify semantic validity against a live database.

The remainder of this paper is organized as follows. Section 2 introduces the Text-to-Flux framework. Section 3 details the proposed TEFD dataset. Performance evaluation is presented in Section

4, results and analysis are discussed in Section 5, related work is reviewed in Section 6, and Section 7 concludes the paper.

2 The FluxEngine Framework

We present **FluxEngine**, an end-to-end framework designed to synthesize high-quality and semantically-grounded (NL, Flux) pairs. The framework is not merely a set of static templates; it is a sophisticated, data-driven generation pipeline that directly interacts with a live InfluxDB instance to ensure the realism and validity of every generated sample.

As illustrated in Figure 1, the framework’s operational logic is divided into two main stages, each comprising one or more internal phases. Stage 1 (Expert Dataset Generation) automatically analyzes raw database statistics to parameterize a diverse set of query templates, generating a technically sound base dataset that is verified for execution correctness through Phase 1 and Phase 2. Stage 2 (Natural Language Augmentation) then builds upon this expert dataset in Phase 3, using a Large Language Model (Gemini 2.5 Pro) to paraphrase the natural language questions into multiple stylistic variations, followed by a crucial consistency verification step to ensure semantic integrity. The following sections detail these phases within the pipeline.

2.1 Phase 1: Live Statistical Profiling

The foundation of our framework is its ability to generate queries that are relevant to the underlying data distribution. To achieve this, the pipeline’s first step is to perform a comprehensive statistical analysis of the live database, a process we call Live Statistical Profiling. For each target measurement (e.g., `airSensors`, `earthquake`), our script (`generate_queries.py`) executes a series of preliminary Flux queries to build a dynamic profile. This profile is critical for data-driven parameterization and includes:

- **Temporal Anchor Extraction**: It identifies the timestamp of the most recent data point (`anchor_time`). This serves as a moving endpoint for generating both absolute and relative time range queries, ensuring all generated time windows contain fresh data.
- **Statistical Moment Calculation**: For each numerical field (e.g., `temperature`, `mag`), it computes key statistics such as the mean and maximum. These values are later used to generate plausible thresholds for filtering queries (e.g., `filter(fn: (r) => r._value > 25.5)`), avoiding arbitrary or out-of-distribution values.
- **Categorical Value Sampling**: For each tag key (e.g., `sensor_id`, `magType`), it samples a representative subset of actual values present in the database. This ensures that filtering conditions on tags (`filter(fn: (r) => r["sensor_id"] == "A01")`) correspond to real entities.

This data-driven approach, which limits its scan to a configurable recent period (e.g., `STATS_RANGE = "-365d"`) for efficiency, is a non-trivial engineering effort that guarantees the semantic realism of our generated dataset.

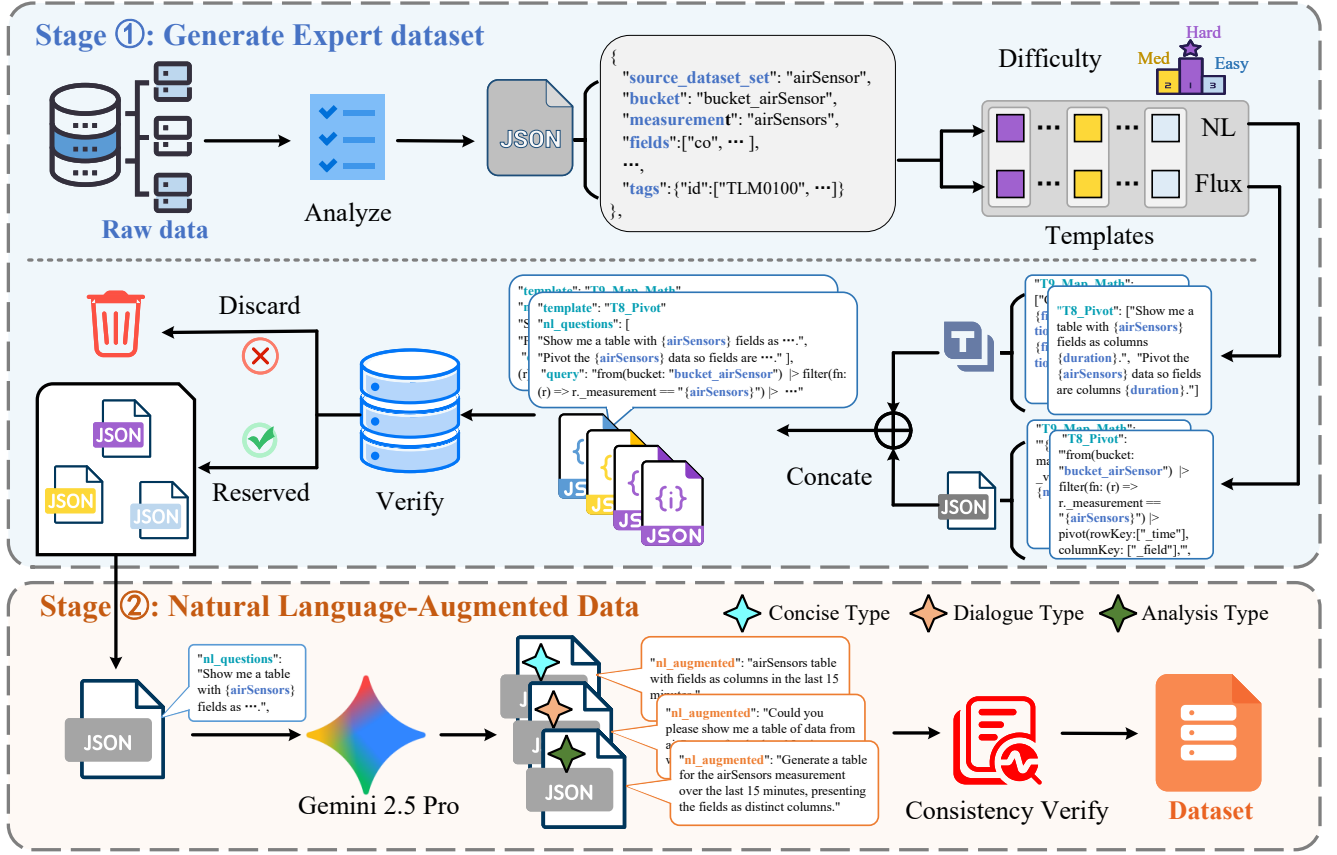


Figure 1: The FluxEngine Framework pipeline, illustrating the two-stage process. Stage 1 Expert Dataset Generation: Raw time-series data is profiled to parameterize query templates. These are used to synthesize and verify a base set of (NL, Flux) pairs of varying difficulty. **Stage 2 Natural Language Augmentation:** The natural language questions from the expert set are paraphrased into diverse stylistic variations (Concise, Dialogue, Analysis) using the Gemini 2.5 Pro model, followed by a strict consistency verification step to produce the final dataset.

2.2 Phase 2: Hierarchical Compositional Synthesis

With the dynamic data profile from Phase 1, the framework enters the core synthesis stage. We employ a hierarchical, compositional generation engine that systematically combines templates and parameters to create a vast and diverse set of queries.

2.2.1 Multi-Axis Combinatorial Expansion. The generation process is an exhaustive combinatorial expansion across multiple axes, ensuring comprehensive coverage of query patterns. The logic iterates through a deeply nested loop structure:

- (1) For each **target dataset** (e.g., `airSensor`, `noaa`).
- (2) For each predefined **time window** (e.g., `'15m'`, `'1h'`, `'7d'`).
- (3) For both **time modes** (relative `'start: -1h'` and absolute `'start: ...'`, `stop: ...'`).
- (4) For each **field** within the measurement.
- (5) For each **tag** and its sampled values.

This systematic approach ensures that we generate a balanced and wide-ranging set of queries, rather than a randomly biased sample.

2.2.2 Template-based Query Construction. At each step of the iteration, the engine selects and populates a suitable template. Our framework defines a rich library of Flux and corresponding NL templates, covering a wide spectrum of operations:

- **Atomic Templates (T-series):** These represent fundamental operations, such as basic filtering (`T5_Tag_Filter`), numerical filtering (`T6_Numeric_Filter`) using the statistically derived thresholds, windowed aggregations (`T4_Windowed_Aggregation`), and advanced transformations like mathematical operations (`T9_Map_Math`) and string matching (`T10_String_Contains`).
- **Composite Templates (C-series):** These model more complex, real-world analytical tasks by composing multiple atomic operations. Examples include finding the time of a maximum value (`C3_Time_of_Max`), which involves a self-join, and joining two distinct data streams based on shared timestamps and tags (`C2_Join_Two_Fields`).

The final output of this phase is a large collection of (NL, Flux) pairs, where each pair is stored with rich metadata, including the template name and the specific components used for its generation.

2.3 Phase 3: Generation with Semantic Safeguards

Instead of a post-facto validation step that simply discards failed queries, our framework integrates semantic safeguards directly into the generation logic. This proactive approach, refined through extensive trial and error, maximizes the yield of valid queries and represents a significant portion of the development workload. Key safeguards implemented in our generator include:

- **Type-Safe Mathematical Operations:** In templates involving arithmetic (`T9_Map_Math`), we explicitly cast values to `float()` before applying operations (e.g., `float(v: r._value) * 1.8 + 32.0`). This prevents runtime errors that commonly occur in Flux when operating on mixed integer and float types.
- **Existence Checks for Robust Filtering:** For templates involving string operations on tag values (`T10_String_Contains`), we prepend an `exists r["tag_key"]` check. This prevents the query from failing on records where the specified tag might not be present, a common scenario in schema-flexible time-series data.
- **Reproducible Randomness:** We utilize a fixed random seed (`random.seed(42)`) for all sampling processes. This ensures that the entire dataset generation process is fully deterministic and reproducible, a cornerstone of reliable scientific research.
- **Realistic Numeric Precision:** All statistically derived thresholds are rounded to two decimal places, mimicking real-world analytical practices and improving the readability of generated queries.

Through this meticulous, multi-phase process—from live data profiling to systematic synthesis and built-in safeguards—the Text2Flux framework produces the TEFD dataset, a benchmark characterized by its scale, diversity, and, most importantly, its semantic fidelity to real-world time-series analysis tasks.

3 The TEFD Dataset

3.1 Data Sources and Domain Coverage

To ensure the semantic richness and cross-domain robustness of TEFD, we orchestrated high-fidelity telemetry data from six distinct real-world domains sourced from the InfluxDB sample repository. These datasets cover a wide spectrum of monitoring scenarios: (i) **Smart Building IoT** (air quality, HVAC); (ii) **Marine Monitoring** (buoy data, wave height); (iii) **Geological Observation** (USGS seismic metrics); (iv) **Industrial IoT** (automated production line); (v) **Wildlife Tracking** (geospatial migration); and (vi) **Hydrology** (water level observations). By incorporating diverse physical metrics and heterogeneous tag structures, TEFD evaluates a model’s ability to perform **precise attribute mapping** and **logic synthesis** across varying domain contexts, moving beyond the repetitive patterns found in single-domain benchmarks.

3.2 Dataset Statistics and Composition

The construction of the TEFD benchmark resulted in a final dataset comprising **8,447** high-quality (NL, Flux) pairs. As illustrated in the comprehensive dashboard (Figure 2), these pairs are synthesized from **1,819** unique base queries, which serve as the logical backbone

for the subsequent natural language augmentation process. A pivotal characteristic of TEFD is its *structural diversity*: our framework utilizes 18 distinct functional templates that cover the full spectrum of time-series analysis tasks, from basic tag-based filtering to advanced composite workflows like temporal period-over-period comparisons and weighted averages (Figure 2a).

The functional richness of TEFD is further quantified by the distribution of Flux operators shown in Figure 3 (Left). While core operators like `range` and `filter` are present in 100% of queries to establish the temporal and logical boundaries of the data stream, the dataset maintains significant coverage of advanced functional operators. Specifically, `aggregateWindow` and `map` are featured in over **45%** of queries, while data-shaping operations like `pivot` and stream-combination operators like `join` or `union` appear in approximately **23%** and **25%** of the samples, respectively. This ensures that TEFD is not merely a retrieval-based benchmark but a rigorous test of a model’s ability to orchestrate complex, state-dependent data transformation pipelines. To formalize this complexity, we categorize templates into three hierarchical levels of analytical depth, as presented in Figure 4.

3.3 Data Split and Template Distribution

To ensure the integrity of the evaluation and assess the model’s true generalization capabilities, we implemented a meticulous **stratified splitting strategy**. Unlike random sampling, which can lead to distribution shifts, we partitioned the 1,819 base queries at the template level before applying natural language augmentation. This ensures that every functional pattern is proportionally represented in the Training (6,725), Validation (788), and Test (934) sets (Figure 3 Right).

The reliability of this strategy is visually confirmed by Figure 2(a), where the near-identical color ratios for each of the 18 templates across splits demonstrate a balanced and unbiased dataset. Furthermore, as detailed in Table 1, we maintained a consistent difficulty distribution across all subsets. Notably, **64.3%** of the benchmark is categorized as **‘Hard’**. This intentional skew toward high-difficulty samples—characterized by deep transformation chains and nuanced temporal logic—is designed to expose the reasoning limits of current Large Language Models in the context of advanced system observability and IoT telemetry.

Table 1: Difficulty Statistics across TEFD Dataset Splits.

Split	Simple (%)	Medium (%)	Hard (%)	Total
Train	380 (5.7%)	2,015 (30.0%)	4,330 (64.4%)	6,725
Val	45 (5.7%)	235 (29.8%)	508 (64.5%)	788
Test	55 (5.9%)	285 (30.5%)	594 (63.6%)	934
Overall	480 (5.7%)	2,535 (30.0%)	5,432 (64.3%)	8,447

3.4 Natural Language Augmentation and Verification

The practical utility of TEFD relies on its linguistic diversity. We utilized a two-stage augmentation pipeline where a Large Language Model paraphrased canonical questions into three distinct styles:

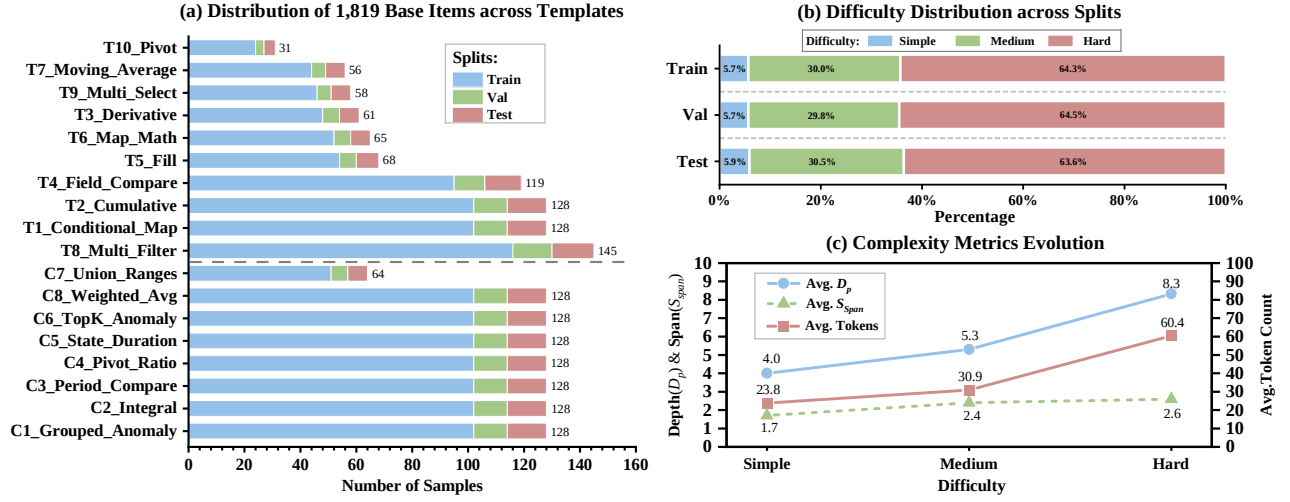


Figure 2: Comprehensive statistical overview of the TEFD benchmark. (a) Distribution of the 1,819 base query items across 18 re-indexed templates. (b) Difficulty distribution across dataset splits, confirming the consistency of our stratified sampling (Hard: ~64.3%). (c) Evolution of complexity metrics: we observe a strong positive correlation between difficulty levels and structural/linguistic complexity (Avg. D_p , S_{span} , and Tokens).

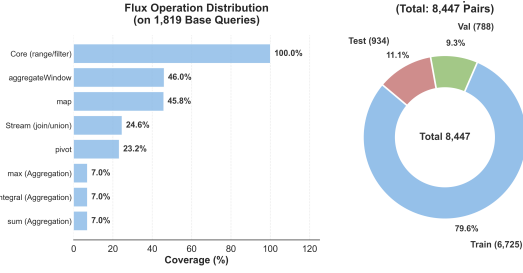


Figure 3: Statistical overview of the TEFD benchmark. (Left) Operator distribution across 1,819 base queries. (Right) Global split ratio for the 8,447 paired samples.

Concise (expert imperative), *Conversational* (novice inquiry), and *Analytical* (formal reportage). To prevent "parameter drift"—where the LLM might erroneously alter timestamps or thresholds—we enforced a strict **Semantic Consistency Verifier** (Algorithm 1). This guardrail ensures that all 8,447 samples maintain perfect numerical and entity alignment with their underlying Flux logic.

3.5 Complexity Analysis

As quantitatively validated in Figure 2(c), TEFD exhibits a strong positive correlation between reasoning difficulty and objective complexity metrics. Unlike declarative SQL benchmarks, Flux necessitates constructing explicit procedural pipelines: the average **Pipeline Depth** (D_p) rises from 4.00 for 'Simple' queries to 8.32 for 'Hard' samples, with a maximum depth of 13. Chaining over eight functional operators requires models to maintain strict state consistency across long, order-sensitive transformation sequences. Simultaneously, linguistic complexity scales significantly, with average **token counts** nearly tripling from 23.8 to 60.4, reflecting

the sophisticated descriptive requirements of advanced time-series tasks. Furthermore, the **Schema Span** (S_{span}) increases from 1.7 to 2.6, demanding precise attribute mapping and coordination across multiple data streams. This multi-dimensional metric escalation establishes TEFD as a rigorous benchmark for evaluating deep procedural logic synthesis.

4 Experimental Evaluation

This section describes our evaluation protocol for Text-to-Flux, including benchmark sampling, compared models, prompting setup, execution environment, and metrics. Our goal is to ensure rigor, reproducibility, and fairness across all evaluated systems.

4.1 Benchmark and Sampling

We evaluate on the TEFD test set ($N=934$). To obtain a cost-effective yet statistically representative evaluation, we perform *stratified sampling* to construct a 200-item subset that preserves the original difficulty distribution: 13 Simple (6.5%), 63 Medium (31.5%), and 124 Hard (62.0%). The sampled subset covers all 18 functional templates in TEFD, ensuring comprehensive testing of the targeted competencies.

4.2 Compared Models

We select five state-of-the-art LLMs spanning both closed-source and open-weight ecosystems.

- **Gemini 2.5 Pro**: Google's flagship model, included for strong procedural reasoning and code generation [24].
- **DeepSeek-V3**: A high-performance open-source Mixture-of-Experts model, representing the frontier of open-weight systems, especially in coding-oriented tasks [4].
- **GPT-4o**: OpenAI's flagship model, widely considered an industry benchmark for complex reasoning [9].

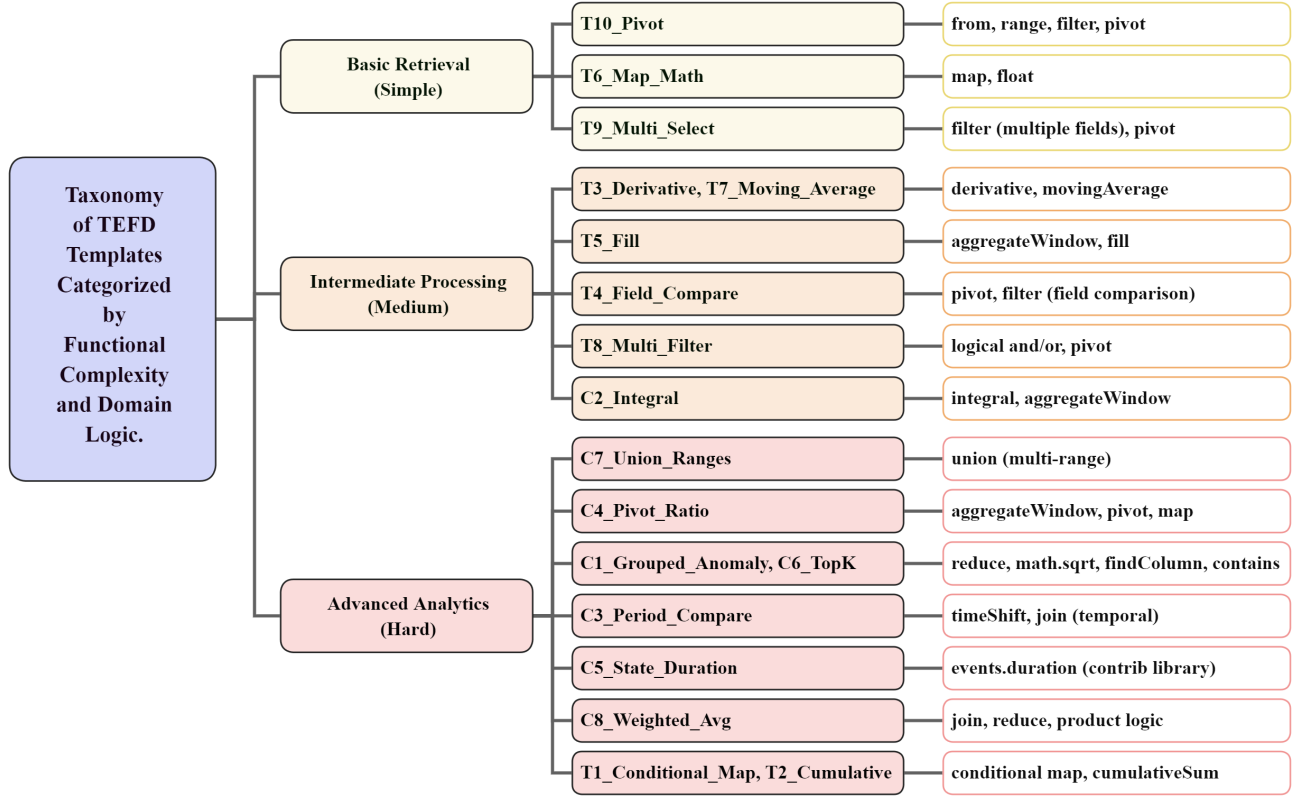


Figure 4: Taxonomy of TEFD Templates Categorized by Functional Complexity and Domain Logic.

Table 2: Benchmark results of five LLMs on the TEFD dataset ($n = 200$). EX: Execution Accuracy. NEA: Non-Empty Accuracy. SynER: Syntax Error Rate. SemER: Semantic Error Rate. TSA-Join: Task-Specific Accuracy for Join operators. Top performance for each metric is in bold.

Model	Overall EX	NEA	SynER	SemER	EX-Simple	EX-Medium	EX-Hard	TSA-Join
Gemini 2.5 Pro	43.50%	39.33%	25.00%	31.50%	100.0%	82.54%	17.74%	5.26%
DeepSeek-V3	42.50%	35.33%	20.50%	37.00%	100.0%	74.60%	20.16%	10.53%
GPT-4o	37.00%	30.67%	27.00%	35.50%	100.0%	60.32%	18.55%	5.26%
Qwen-Max	34.50%	26.00%	30.50%	34.50%	84.62%	58.73%	16.94%	10.53%
GPT-4o-mini	19.00%	13.33%	62.00%	19.00%	30.77%	41.27%	6.45%	0.00%

Table 3: Task-Specific Accuracy (TSA) across five functional dimensions. All values represent Execution Accuracy (EX) on samples containing specific operator categories.

Model	Window	Temp.	Math	Trans.	Logic
Gemini 2.5 Pro	7.87%	32.65%	23.76%	31.30%	39.29%
DeepSeek-V3	10.11%	36.73%	25.74%	36.64%	41.07%
GPT-4o	5.62%	42.86%	26.73%	21.37%	27.68%
Qwen-Max	4.49%	32.65%	22.77%	24.43%	29.46%
GPT-4o-mini	2.25%	24.49%	7.92%	16.03%	17.86%

- **GPT-4o-mini**: A lightweight GPT-4o variant to assess the performance–efficiency trade-off under reduced capacity [9].

- **Qwen-Max**: A leading large-scale model from Alibaba, included to diversify model ecosystems [2].

4.3 Prompting and Decoding Configuration

All models are evaluated under a strict **zero-shot** setting without task-specific fine-tuning. The standardized prompt provides (i) the database schema, (ii) the natural language query, and (iii) an explicit instruction to output *only* an executable Flux program (no explanations). To ensure deterministic and reproducible decoding, we set the temperature to 0.0 for all models.

4.4 Execution Environment and Temporal Normalization

The evaluation pipeline is fully automated. Each generated Flux query is executed against a locally hosted **InfluxDB v2.3.0+SNAPSHOT** instance. A key challenge in TSDB evaluation is the temporal dependency of relative-time expressions (e.g., `range(start: -1h)`), which can yield different outputs as wall-clock time advances.

To ensure fair and reproducible evaluation, we implement a custom *QueryPatcher* module. Before execution, it parses each predicted Flux script and deterministically rewrites all relative time ranges into absolute timestamps anchored to a fixed evaluation time. As a result, all model-generated queries are executed against the same effective data snapshot, enabling apples-to-apples comparison. Execution outputs are then programmatically compared with the gold outputs to compute scores.

4.5 Evaluation Metrics

Surface-form matching is unreliable for the Text-to-Flux task: semantically equivalent Flux pipelines can differ significantly in syntax, while minor procedural variations—such as operator ordering, window boundary definitions, or type casting—can substantially alter the resulting data stream. Following established execution-based evaluation paradigms in semantic parsing and NL2SQL [13, 27], we evaluate model predictions by executing them against a live database instance and comparing the resulting output streams against ground-truth references.

Holistic Effectiveness Metrics. We employ two primary metrics to measure end-to-end success. **Execution Accuracy (EX)** serves as our gold standard, measuring functional equivalence under execution. A prediction q_{pred} is considered correct if its result stream $\mathcal{R}(q_{pred})$ exactly matches the gold stream $\mathcal{R}(q_{gold})$ across timestamps, fields, and tag dimensions:

$$EX = \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \mathbb{I}(\mathcal{R}(q_{pred}^{(i)}) = \mathcal{R}(q_{gold}^{(i)})) \quad (1)$$

Furthermore, to mitigate potential bias from "empty matches"—where an incorrect query might coincidentally return an empty stream identical to the gold query due to data sparsity—we report **Non-Empty Accuracy (NEA)**. This metric conditions the evaluation strictly on instances where the gold execution result is non-empty [18]:

$$NEA = \frac{\sum \mathbb{I}(\mathcal{R}(q_{pred}) = \mathcal{R}(q_{gold}) \wedge \mathcal{R}(q_{gold}) \neq \emptyset)}{\sum \mathbb{I}(\mathcal{R}(q_{gold}) \neq \emptyset)} \quad (2)$$

Diagnostic Error Metrics. To systematically analyze model failures, we categorize incorrect predictions into two diagnostic classes. The **Syntax Error Rate (SynER)** measures the fraction of outputs that fail to execute due to Flux grammar violations, invalid import paths, or runtime type mismatches. Conversely, the **Semantic Error Rate (SemER)** captures instances where the query executes successfully but produces an incorrect result stream, reflecting logical failures such as inappropriate aggregation choices or misordered functional transformations.

Task-Specific Accuracy (TSA). To localize performance bottlenecks within specific operator families, we introduce **TSA**, which computes execution correctness on subsets of queries containing

specific operator classes C (e.g., windowing, joins, or mathematical transformations):

$$TSA_C = \frac{\sum_{q \in \mathcal{D}} \mathbb{I}(\text{Op}(q_{gold}, C) \wedge \text{Correct}(q_{pred}))}{\sum_{q \in \mathcal{D}} \mathbb{I}(\text{Op}(q_{gold}, C))} \quad (3)$$

We specifically highlight **TSA-Join**, as the procedural orchestration of multiple streams represents a core reasoning challenge where current LLMs frequently fail to maintain temporal and schema consistency.

5 Results and Analysis

5.1 Overall Results

Table 2 reports the overall performance of all baselines on our 200-sample benchmark. Gemini 2.5 Pro achieves the best overall EX (43.50%) and NEA (39.33%), while DeepSeek-V3 is highly competitive (42.50% EX) and attains the lowest SynER (20.50%). GPT-4o reaches 37.00% EX, and Qwen-Max obtains 34.50% EX. GPT-4o-mini substantially underperforms (19.00% EX) and exhibits a high SynER (62.00%), indicating frequent execution failures under constrained capacity.

5.2 Results by Difficulty

All large models achieve near-perfect performance on Simple queries (100% for Gemini 2.5 Pro, DeepSeek-V3, and GPT-4o). However, performance drops sharply on Hard queries: top models reach only **17–20% EX**, revealing a strong performance cliff as pipelines become deeper and more order-sensitive. This gap suggests that while models can handle shallow retrieval and basic transformations, they struggle to reliably compose long Flux pipelines that require precise intermediate schema states and step ordering.

5.3 Error Breakdown: Syntax vs. Semantics

A key observation is the *syntax vs. semantics* gap. DeepSeek-V3 achieves the lowest SynER (20.50%), indicating stronger executable fluency; yet its SemER remains high (37.00%), implying that many executable programs still implement incorrect transformation logic. This pattern suggests that execution validity does not guarantee semantic correctness in Flux, where subtle operator misordering (e.g., missing `aggregateWindow` before `cumulativeSum`) can produce plausible but incorrect streams.

5.4 Functional Dimensional Analysis (TSA)

Overall EX obscures fine-grained functional strengths and weaknesses. Prior work suggests temporal and logical reasoning should be evaluated as distinct dimensions, especially when semantics depend on time-dependent operators and logical composition [22]. We therefore decompose evaluation into five TSA dimensions (Window, Temporal, Math, Trans., Logic), summarized in Table 3.

Key Findings. (i) **Windowing bottleneck.** TSA-Window is consistently low across all models (no model exceeds 10.11%), indicating that composing correct windowing/resampling chains remains highly challenging in Flux. (ii) **Divergent strengths.** GPT-4o peaks on TSA-Temp (42.86%), while Gemini 2.5 Pro and DeepSeek-V3 are stronger on Trans./Logic, suggesting better schema manipulation

consistency. **(iii) Scale effect.** GPT-4o-mini underperforms across all dimensions, supporting that longer procedural pipelines amplify errors for smaller models.

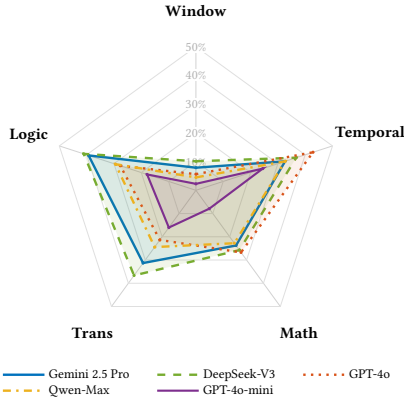


Figure 5: Capability fingerprints of five LLMs across functional TSA dimensions. The plot highlights consistently low performance on Windowing and divergent strengths across Temporal vs. structural reasoning.

5.5 Discussion

The results reveal three consistent limitations of current LLMs on Text-to-Flux:

Performance cliff on complex pipelines. Accuracy drops sharply from near-perfect on Simple queries to only 17–20% on Hard queries, confirming that deep, order-sensitive Flux pipelines remain a major bottleneck.

Syntax vs. semantics gap. DeepSeek-V3 achieves the lowest SynER (20.50%), yet SemER remains high across leading models (31.50%–37.00%), indicating that executable outputs often implement the wrong transformation sequence (e.g., missing `aggregateWindow` before `cumulativeSum`).

Multi-stream orchestration weakness. **TSA-Join** is extremely low (best 10.53%), suggesting models struggle with Flux’s procedural stream alignment (join keys, schema consistency) compared to SQL’s declarative JOIN.

Open-weight competitiveness. DeepSeek-V3 is on par with Gemini 2.5 Pro overall and slightly stronger on Hard queries, suggesting TEFD can serve as a sensitive benchmark for tracking progress of open-weight models on procedural time-series pipelines.

6 Related Work

Our work is closely related to the research areas of text-to-query benchmarking, NLI for monitoring and time-series, synthetic data and self-onstruct logic, and LLMs in time-series analysis.

6.1 Text-to-Query Benchmarking

Text-to-Query evaluation has shifted from academic structural generalization to industrial robustness. Early benchmarks such as WikisQL and Spider [27] emphasized cross-domain schema linking,

while later efforts demonstrated that strong performance on clean, static snapshots often fails to transfer to production-like settings. BIRD [13] introduced large-scale *dirty* databases and efficiency-aware evaluation, and Spider 2.0 [12] further escalated the setting toward enterprise workflows in modern data warehouses, reinforcing *execution-based* evaluation as the practical criterion.

As the community expands beyond relational SQL, TEND [16] targets NL-to-NoSQL with nested structures. In parallel, our benchmark focuses on Time-Series Databases (TSDBs), where queries operate over continuous streams and require procedural composition over functional operators. Compared to relational snapshots, Text-to-Flux introduces domain-specific challenges such as multi-scale resampling and temporally aligned stream composition, motivating TEFD as a dedicated benchmark for *analysis-pipeline* generation in Flux.

6.2 NLI for Monitoring and Time-Series

LLM-driven NLI for observability typically cover metrics DSLs (e.g., PromQL), logs DSLs (e.g., LogQL), and time-series SQL. PromQL generation has been studied via structured systems for query construction [28], while NL2QL targets log analytics with dedicated paired data for Loki queries [20]. However, existing monitoring NLI are often limited to relatively shallow expressions [20, 28]: a recent survey highlights difficulties in composing long, state-dependent programs and reliably handling complex analytical intents [15].

Flux differs materially from declarative PromQL/SQL: it is a functional, pipelined scripting language where operator ordering and intermediate table schemas are critical (e.g., `aggregateWindow` → `derivative` → `movingAverage`). Although industrial tooling has begun exploring Flux conversion [10], the research community lacks a verified benchmark that supports execution-based evaluation under time-dependent semantics. TEFD fills this gap by focusing on deep pipeline synthesis and temporal validity in TSDB querying.

6.3 Synthetic Data and Self-Instruct Logic

Recent work shows that high-quality synthetic code data and automatic verification can substantially improve specialized LLMs. Magicoder [25] distills instruction data from open-source code, while OpenCodeInstruct [1] and Case2Code [21] emphasize execution feedback for semantic fidelity. FluxSynth follows this line by using a live database as an execution oracle, aligning generation with verifiable time-series semantics and reducing parameter drift during augmentation [5].

6.4 LLMs in Time-Series Analysis

LLMs are increasingly used for time-series reasoning and diagnosis beyond query translation [11], including agentic systems for detection and causal analysis such as Argos [8] and Time-RA [26]. TEFD complements these directions by providing a benchmarked NL-to-DSL execution layer for TSDB analytics, enabling rigorous evaluation of the query/program synthesis component in end-to-end observability agents.

Unlike relational benchmarks that rely on static snapshots, TEFD is designed to handle the functional orchestration and temporal

decay characteristic of TSDB querying. We move beyond the shallow retrieval patterns seen in prior monitoring NLIs to focus on the procedural synthesis of deep operator chains. A key methodological distinction is the introduction of a Self-Sustaining Live Context and the QueryPatcher utility; together, they guarantee the long-term reproducibility of execution results involving relative time ranges. This infrastructure allows TEFD to serve as a verified, execution-based benchmark that specifically targets the reasoning bottlenecks found in real-world time-series analysis.

7 Conclusion

This work presents **TEFD**, the first large-scale benchmark for Text-to-Flux, along with **FluxEngine**, an automated construction pipeline that addresses temporal staleness and functional pipeline complexity in time-series query generation. By maintaining a self-sustaining live data context, Text2Flux ensures that relative-time queries remain executable as time advances, making execution-based evaluation feasible in settings where static snapshots quickly lose validity.

Although TEFD establishes a reproducible evaluation substrate for Text-to-Flux, a number of challenges remain. Extending the benchmark to cover more complex stateful operators, long-horizon dependencies, and multi-stream joins is particularly important, as practical time-series analytics frequently involve heterogeneous data streams with misaligned temporal semantics. Beyond benchmarking, future work may also consider interactive and multi-turn Text-to-Flux scenarios, as well as the use of TEFD as a training or alignment signal, to support more robust and semantically grounded natural language interfaces for time-series databases.

Acknowledgments

Haodi Zhang and Yuanfeng Song are the corresponding authors of this paper. We are grateful to the anonymous reviewers for their constructive comments on this paper.

References

- [1] Wasi Uddin Ahmad, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Vahid Noroozi, Somshubra Majumdar, and Boris Ginsburg. 2025. OpenCodeInstruct: A Large-scale Instruction Tuning Dataset for Code LLMs. *arXiv preprint arXiv:2504.04030* (2025).
- [2] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609* (2023).
- [3] Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Haotian Wang, Ming Liu, and Bing Qin. 2024. TimeBench: A Comprehensive Evaluation of Temporal Reasoning Abilities in Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL)*.
- [4] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojuan Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Pei Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qishi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shutong Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanbiao Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinghai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yudian Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] <https://arxiv.org/abs/2412.19437>
- [5] Quanting Dong, Keming Lu, Chengpeng Li, Tingyu Xia, Bowen Yu, Chang Zhou, and Jingren Zhou. 2024. Self-play with Execution Feedback: Improving Instruction-following Capabilities of Large Language Models. arXiv:2406.13542 [cs.CL] <https://arxiv.org/abs/2406.13542>
- [6] Bahare Fatemi, Seyed Mehran Kazemi, Anton Tsitsulin, Karishma Malkan, Jinyeong Lim, John Palowitch, Sungyong Seo, Jonathan Halcrow, and Bryan Perazzi. 2025. Test of time: A benchmark for evaluating llms on temporal reasoning. In *International Conference on Learning Representations*, Vol. 2025. 94426–94447.
- [7] Thomas R. G. Green and Marian Petre. 1996. Usability analysis of visual programming environments: a ‘cognitive dimensions’ framework. *Journal of Visual Languages & Computing* 7, 2 (1996), 131–174.
- [8] Yile Gu, Yifan Xiong, Jonathan Mace, Yuting Jiang, Yigong Hu, Baris Kasikci, and Peng Cheng. 2025. Argos: Agentic time-series anomaly detection with autonomous rule generation via large language models. *arXiv preprint arXiv:2501.14170* (2025).
- [9] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276* (2024).
- [10] InfluxData. 2025. *Introducing the InfluxDB 3 MCP Server: Natural Language for Time Series Data*. <https://www.influxdata.com/blog/influxdb-mcp-server/>
- [11] Yushan Jiang, Kanghui Ning, Zijie Pan, Xuyang Shen, Jingchao Ni, Wenchao Yu, Anderson Schneider, Haifeng Chen, Yuriy Nevmyvaka, and Dongjin Song. 2025. Multi-modal time series analysis: A tutorial and survey. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*. 6043–6053.
- [12] Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, Victor Zhong, Caiming Xiong, Ruoxi Sun, Qian Liu, Sida Wang, and Tao Yu. 2024. Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. arXiv:2411.07763 [cs.CL] <https://arxiv.org/abs/2411.07763>
- [13] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? *Proceedings of the VLDB Endowment* 17, 11 (2024).
- [14] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM already serve as a database interface? a big bench for large-scale database grounded text-to-SQLs. In *Proceedings of the 37th International Conference on Neural Information Processing Systems (New Orleans, LA, USA) (NIPS ’23)*. Curran Associates Inc., Red Hook, NY, USA, Article 1835, 28 pages.
- [15] M. Liu and J. Xu. 2025. NLI4DB: A Systematic Review of Natural Language Interfaces for Databases. *arXiv preprint arXiv:2503.02435* (2025).
- [16] Jinwei Lu, Yuanfeng Song, Zhiqian Qin, Haodi Zhang, Chen Zhang, and Raymond Chi-Wing Wong. 2025. Bridging the gap: Enabling natural language queries for nosql databases through text-to-nosql translation. *arXiv preprint arXiv:2502.11201* (2025).
- [17] Leo A. Meyerovich and Ariel Rabkin. 2013. Empirical Analysis of Programming Language Adoption. In *Proceedings of the 2013 ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*.
- [18] Anton Osokin, Irina Saparina, and Ramil Yarullin. 2023. Searching for Better Database Queries in the Outputs of Semantic Parsers. In *Findings of the Association for Computational Linguistics: EACL 2023*, Andreas Vlachos and Isabelle Augenstein (Eds.). Association for Computational Linguistics, Dubrovnik, Croatia, 2243–2256. doi:10.18653/v1/2023.findings-eacl.168
- [19] Zhiqian Qin, Yuanfeng Song, Jinwei Lu, Yuanwei Song, Shuaimin Li, and Chen Jason Zhang. 2025. MultiTEND: A Multilingual Benchmark for Natural Language to NoSQL Query Translation. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 – August 1, 2025*. Association for Computational Linguistics, 24632–24657.
- [20] Vishwanath Seshagiri, Siddharth Balyan, Vaastav Anand, Kaustubh Dhole, Ishan Sharma, Avani Wildani, José Cambroner, and Andreas Züfle. 2024. Chatting

with logs: An exploratory study on finetuning llms for logql. *arXiv preprint arXiv:2412.03612* (2024).

- [21] Yunfan Shao, Linyang Li, Yichuan Ma, Peiji Li, Demin Song, Qinyuan Cheng, Shimin Li, Xiaonan Li, Pengyu Wang, Qipeng Guo, et al. 2025. Case2Code: Scalable synthetic data for code generation. In *Proceedings of the 31st International Conference on Computational Linguistics*. 11056–11069.
- [22] Wen Tang and Vaishak Belle. 2024. LTLBench: Towards benchmarks for evaluating temporal logic reasoning in large language models. *arXiv:2407.05434 [cs.CL]*
- [23] Xinye Tang, Amir H Abdi, Jeremias Eichelbaum, Mahan Das, Alex Klein, Nihal Irmak Pakis, William Blum, Daniel L Mace, Tanvi Raja, Namrata Padmanabhan, et al. 2024. NL2kql: From natural language to kusto query. *arXiv preprint arXiv:2404.02933* (2024).
- [24] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805* (2023).
- [25] Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2024. Magicoder: empowering code generation with OSS-INSTRUCT. In *Proceedings of the 41st International Conference on Machine Learning (Vienna, Austria) (ICML'24)*. JMLR.org, Article 2158, 26 pages.
- [26] Yiyuan Yang, Zichuan Liu, Lei Song, Kai Ying, Zhiguang Wang, Tom Bamford, Svitlana Vyetenko, Jiang Bian, and Qingsong Wen. 2025. Time-ra: Towards time series reasoning for anomaly with llm feedback. *arXiv preprint arXiv:2507.15066* (2025).
- [27] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 conference on empirical methods in natural language processing*. 3911–3921.
- [28] Chenxi Zhang, Bicheng Zhang, Dingyu Yang, Xin Peng, Miao Chen, Senyu Xie, Gang Chen, Wei Bi, and Wei Li. 2026. PromCopilot: Simplifying Prometheus Metric Querying in Cloud Native Online Service Systems via Large Language Models. *ACM Transactions on Software Engineering and Methodology* (2026).
- [29] Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Aiwei Liu, Yong Yang, Zhonghai Wu, Xuming Hu, Philip Yu, and Ying Li. 2025. A survey of aiops in the era of large language models. *Comput. Surveys* 58, 2 (2025), 1–35.
- [30] Zihao Zhou and Rose Yu. 2025. Can LLMs understand time series anomalies?. In *International Conference on Learning Representations (ICLR)*.

A Detailed Dataset Complexity Profile

Table 4 provides a granular breakdown of the structural and linguistic complexity metrics across TEFD partitions. The data highlights three critical aspects of our benchmark:

- **Procedural Depth Expansion:** Unlike declarative SQL, Flux requires constructing explicit transformation pipelines. The average **Pipeline Depth** (D_p) scales from 4.00 in *Simple* queries to **8.32** in *Hard* samples, with the most complex analytics reaching a depth of **13** functional operators.
- **Linguistic and Schema Pressure:** As reasoning difficulty increases, the average **Token Count** nearly triples ($23.8 \rightarrow 60.4$), and the **Schema Span** (S_{span}) grows from 1.7 to 2.6. This forces models to handle both sophisticated verbal descriptions and precise attribute mapping across heterogeneous data streams.
- **Split Consistency:** The metrics demonstrate high statistical stability across the *Train*, *Val*, and *Test* sets (e.g., *Hard* D_p remains consistent at ~ 8.32). This verifies that FLUXENGINE successfully preserves the underlying complexity distribution through its stratified sampling strategy.

Table 4: Detailed Complexity Metrics of TEFD across Difficulty Levels and Splits.

Split	Diff.	Count	Avg D_p	Max D_p	Tokens	S_{span}
Train	Simple	380	4.00	4	23.8	1.7
	Med.	2015	5.30	6	30.9	2.4
	Hard	4330	8.32	13	60.4	2.6
Val	Simple	45	4.00	4	22.6	1.7
	Med.	235	5.32	6	31.0	2.4
	Hard	508	8.31	13	60.2	2.6
Test	Simple	55	4.00	4	22.9	1.6
	Med.	285	5.25	6	30.1	2.4
	Hard	594	8.32	13	60.4	2.6
Overall	–	8447	6.91	13	48.1	2.4

B Extended Task Illustration

To provide a concrete understanding of the Text-to-Flux task, Figure 6 illustrates a representative sample from the TEFD dataset. Unlike the relational patterns in Text-to-SQL, this task requires orchestrating a functional pipeline where the order of operations—such as range selection before aggregateWindow—is semantically critical.

Furthermore, the illustration highlights the necessity of the **Self-Sustaining Live Context**. In traditional static benchmarks, a query targeting the "past 1 hour" would return null results shortly after the dataset’s creation. Our framework solves this *temporal obsolescence* by perpetually injecting fresh telemetry, ensuring that the execution-based evaluation remains deterministic and semantically grounded over indefinite time horizons.

C Detailed Template Taxonomy

The structural diversity of TEFD is underpinned by 18 re-indexed functional templates. As summarized in Table 5, these templates categorize analytical intents into three hierarchical tiers:

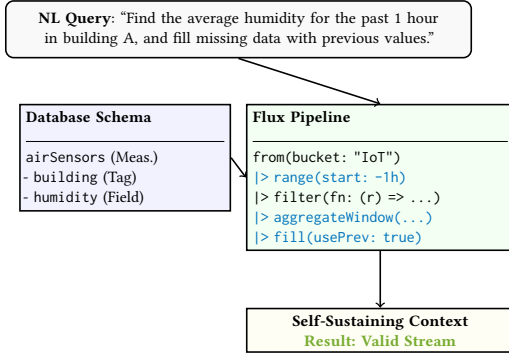


Figure 6: Illustration of the Text-to-Flux task. The pipeline requires orchestrating multiple functional operators while remaining anchored to a live data context.

- **Basic Tier:** Exercises fundamental stream selection and arithmetic casting, representing ~5.7% of the benchmark.
- **Intermediate Tier:** Focuses on single-stream analytical functions like derivative and movingAverage, making up ~30.0% of samples.
- **Advanced Tier:** Targets the reasoning limits of LLMs by requiring the coordination of multi-stream joins, multi-range unions, and complex state-dependent operations like cumulativeSum, representing the majority (64.3%) of the dataset.

D Natural Language Augmentation and Verification

To transform technical templates into natural human intents, we utilized a two-stage LLM-based augmentation pipeline. However, LLMs are prone to *parameter drift*—accidentally altering timestamps or sensor IDs during paraphrasing.

To mitigate this, we implemented the **Consistency Verifier** (Algorithm 1). This module treats the query metadata as immutable facts and programmatically scans the LLM output to ensure all numerical thresholds and schema entities are preserved verbatim. Only variations that maintain 100% literal consistency with the underlying Flux logic are committed to the final 8,447 paired samples.

Algorithm 1 Semantic Consistency Verification

Require: C (Metadata), S_{orig} (Original NL), S_{gen} (Generated NL)

Ensure: is_valid

```

1:  $N_{orig} \leftarrow \text{ExtractNumericLiterals}(S_{orig})$ 
2:  $N_{gen} \leftarrow \text{ExtractNumericLiterals}(S_{gen})$ 
3: if  $N_{orig} \not\subseteq N_{gen}$  then
4:   return False
5: end if
6: for all  $entity \in \{\text{measurement, field, tag}\}$  do
7:    $v \leftarrow \text{GetValue}(C, entity)$ 
8:   if  $v \in S_{orig} \wedge v \notin S_{gen}$  then
9:     return False
10:  end if
11: end for
12: return True
  
```

E Predominant Error Archetypes

To better understand the limitations of current LLMs in the Text-to-Flux task, we conducted a manual audit of execution failures. We categorize the predominant errors into three archetypes: *Library Fragility*, *Dynamic Over-engineering*, and *Operator Hallucination*.

E.1 Error Taxonomy

Table 6 summarizes the typical pitfalls encountered by the models. While models exhibit strong natural language understanding, they struggle with the strict syntactic and environmental requirements of the Flux functional engine.

Table 6: Taxonomy of common execution errors in TEFD.

Error Category	Description
Library Fragility	Incorrect or obsolete import paths for community/contrib packages (e.g., contrib/anaisdg).
Over-engineering	Attempting complex dynamic scripting (variables, math) where static hardcoding is required for execution.
Operator Hallucination	Inventing intuitive but non-existent functions (e.g., weightedAverage) instead of verbose functional chains.

E.2 Representative Case Studies

E.2.1 Library Path Hallucination (Case #228). As shown in Table 7, the model correctly identifies the need for an external library to handle events.duration. However, it hallucinates a plausible but invalid path (contrib/anaisdg/events). This highlights the *syntactic fragility* of Flux; unlike SQL, Flux is highly sensitive to the evolution of its package ecosystem, and models often suffer from outdated or fragmented memory of these paths.

E.2.2 Dynamic Logic vs. Static Reality (Case #754). Case #754 (Table 8) reveals a gap in *procedural reasoning*. The model attempts to be “smart” by calculating the duration between two timestamps and dynamically shifting the range. While this reflects superior logical reasoning, it fails in execution because the model utilizes the experimental library with an incorrect path. The Gold Query avoids this by pre-computing static timestamps, suggesting that for Flux NLIs, *conservative generation* is often more robust than *dynamic scripting*.

E.2.3 The Complexity Gap: Weighted Average (Case #692). Case #692 is perhaps the most illustrative of the Text-to-Flux challenge. To calculate a weighted average, the model assumes the existence of a weightedAverage() operator. This is a reasonable assumption in most data processing languages but is **false** in standard Flux.

As illustrated in the Gold Query, Flux requires a verbose 8-line pipeline involving join, reduce, and multiple map transformations. This *Operator Hallucination* demonstrates that models rely on analogies from SQL or Python, failing to recognize that Flux often requires decomposing high-level concepts into low-level stream transformations.

Table 5: Taxonomy of TEFD Templates Categorized by Functional Complexity and Domain Logic.

Category	Templates	Core Flux Operators	Difficulty
Basic Retrieval	T10_Pivot	from, range, filter, pivot	Simple
	T6_Map_Math	map, float	Simple
	T9_Multi_Select	filter (multiple fields), pivot	Simple
Intermediate Processing	T3_Derivative, T7_Moving_Average	derivative, movingAverage	Medium
	T5_Fill	aggregateWindow, fill	Medium
	T4_Field_Compare	pivot, filter (field comparison)	Medium
	T8_Multi_Filter	logical and/or, pivot	Medium
	C2_Integral	integral, aggregateWindow	Medium
Advanced Analytics	C7_Union_Ranges	union (multi-range)	Hard
	C4_Pivot_Ratio	aggregateWindow, pivot, map	Hard
	C1_Grouped_Anomaly, C6_TopK	reduce, math.sqrt, findColumn, contains	Hard
	C3_Period_Compare	timeShift, join (temporal)	Hard
	C5_State_Duration	events.duration (contrib library)	Hard
	C8_Weighted_Avg	join, reduce, product logic	Hard
	T1_Conditional_Map, T2_Cumulative	conditional map, cumulativeSum	Hard

Table 7: Comparison of correct and hallucinated library paths.

Case ID: 228	Library Path Fragility
NL Query	Identify periods where wind speed remained above 7.7 for > 900s.
Gold Query	import "contrib/tomhollingworth/events" ...
Pred Query	import "contrib/anaisdg/events" ...
Error	invalid import path contrib/anaisdg/events

Table 8: Analysis of reasoning-heavy but execution-failed queries.

Case ID: 754	Over-engineering Failure
Strategy	Pred: Dynamic subDuration; Gold: Static range.
Insight	Logic is correct, but execution fails due to library dependencies.

E.3 Summary of Insights

Our qualitative analysis suggests that:

- **Syntactic Correctness ≠ Execution Validity:** Models often fail on the “last mile” of execution (imports/types).
- **Need for Specialized Fine-tuning:** General-purpose LLMs struggle with the functional verbosity of Flux, emphasizing the value of our **TEFD** dataset for teaching these specific procedural chains.